DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

Implementación de un algoritmo de seguimiento de trayectorias en el framework ROS

AUTOR: JUAN FCO. RASCÓN CRESPO

PROFESOR: LUIS E. MORENO LORENTE

7 de Abril de 2014

Índice general

Lista de Figuras

Capítulo 1

Introducción

1.1. Objetivos

Un robot completamente autónomo debe ser capaz de obtener e interpretar la información del entorno para ejecutar diversas tareas. El problema de la navegación y localización de un robot móvil en un entorno está estrechamente relacionado con estas habilidades. Este proyecto final de carrera es la continuación lógica del trabajo realizado en la tesis de máster [1].

El proyecto actual surgió de la necesidad de dotar al robot manipulador Manfred de la habilidad de desplazarse de manera autónoma por un entorno real haciendo uso únicamente de la información que puede captar del entorno a través de sus sensores y de un mapa de dicho entorno. Este mapa a su vez es construido usando únicamente la información recolectada por el telémetro láser del robot y por la información de los desplazamientos de las ruedas de la base.

El robot manipulador Manfred es una de las varias plataformas robóticas con las que cuenta grupo de investigación Robotics Lab, perteneciente al Departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid. Todas las plataformas disponibles han sido diseñadas y construidas por el propio grupo.

El cometido de este proyecto es doble. En primer lugar desarrollar un algoritmo de seguimiento de trayectorias basado en el conocido método de navegación pure-pursuit. En segundo lugar integrar este algoritmo y el resto de algoritmos disponibles para la plataforma róbotica Manfred en el framework de control `ROS`. Por tanto será necesario investigar todas las alternativas que nos ofrece el framework para desarrollar una arquitectura de control que permita comunicar y coordinar a todos los procesos que se deben ejecutar en el robot.

De los múltiples frameworks disponibles en la actualidad que permiten construir una arquitectura de control para robots se eligió `ROS`, por ser un framework al alza, avalado por una gran comunidad de investigadores y desarrolladores, que dispone de muchas implementaciones de algoritmos, listas para ser usadas en tu robot.

1.2. Estado del arte

En esta sección se realiza una revisión del estado del arte de los robots móviles, su evolución, las configuraciones más empleadas, los actuadores utilizados habitualmente y algunas técnicas de control que se han aplicado con el objetivo de conseguir cada vez mayor autonomía en los problemas de seguimiento de trayectoria y en la evasión de obstáculos.

La robótica ha jugado un papel importante en el desarrollo de la humanidad y su evolución ha ido siempre ligada con la construcción de artefactos que faciliten el trabajo. En antiguas civilizaciones, como la griega, se hablaba de seres mecánicos con vida movidos por mecanismos construidos con poleas y bombas hidráulicas. En la civilización árabe se utilizaban artefactos de este tipo para dar confort al ser humano. Con el paso del tiempo se fueron desarrollando multitud de figuras dotadas de partes móviles aunque por aquel entonces no se tenía un concepto general de cómo definirlas.

En 1920 el escritor checo Capek, en su obra dramática Rossum's Universal Robots, acuñó el término robot, a partir de la palabra checa robota, que significa servidumbre o trabajo forzado. Asimov introdujo por primera vez el término robótica en la historia Runaround (círculo vicioso en inglés) que escribió en 1942. En ese mismo relato también aparecieron por vez primera las famosas tres leyes de la robótica. Como consecuencia de la aparición de la computadora y de la mejora de la tecnología de circuitos impresos se pudieron desarrollar los primeros intentos de creación de un verdadero robot en la década de los cuarenta. En el MIT en 1952, aparece la primera máquina de control numérico capaz automatizar algunas tareas industriales. Por su parte, la compañía Unimates, introdujo el primer robot industrial en la General Motors en 1961.

Con la venida de nuevas tecnologías de planificación y razonamiento automático, de 1966 a 1972 se desarrolló en el SRI el primer robot móvil llamado Shakey, que era una plataforma móvil independiente controlada por visión mediante una cámara y dotada con un detector táctil. A partir de ese momento, la investigación y diseño de robots móviles creció de manera exponencial. En 1977 en el Jet Propulsion Laboratory (JPL) se desarrolló el Lunar rover, un robot diseñado

particularmente para la exploración planetaria. A finales de esa década, Moravec desarrolló el robot Stanford cart, capaz de seguir una trayectoria delimitada por una línea establecida en una superficie. En 1983, el robot Raibert fue desarrollado en el MIT. Este robot contaba con una sola pata. Fue diseñado para estudiar la estabilidad de éstos sistemas. A principios de la década del noventa se desarrolló en el MIT un robot uniciclo, es decir, de una sola rueda.

Años más tarde, en 1994, el instituto de robótica CMU desarrolló el robot Dante II, un sistema de seis patas. En 1996 también en el CMU, se desarrolló el robot Gyrover, un robot de una sola rueda estabilizado mediante un giroscopio. Ese mismo año se desarrolló en el MIT el Spring Flamingo, un robot que emulaba el movimiento de un flamingo. Por su parte, la NASA en 1997 envío a Marte un robot móvil teleoperado llamado Sojourner, dedicado a enviar fotografías del entorno de dicho planeta. Ese mismo año, la empresa japonesa HONDA, dio a conocer el robot P3, el primer humanoide capaz de imitar movimientos del cuerpo humano. Al siguiente año, se desarrolla en la universidad Waseda en Japón, el WABIAN R-III, un robot humanoide. En 1999 en el CMU, Zeglin propuso un nuevo diseño de robot con una pata llamado Bow Leg Hopper, un diseño que permite almacenar la energía potencial de la pata. En 2006 Hollis desarrolla el robot Ballbot, un sistema holónomo cuyo movimiento es proporcionado por una sola esfera ubicada en la parte inferior de la estructura. Sin embargo, el estudio de este tipo de robots con una esfera, fue iniciado por Koshiyama y Yamafuji en 1991. En 2009 los rover de la NASA Spirit y Opportunity fueron enviados a Marte para explorar la superficie del planeta en busca de mantos acuíferos. La exploración del planeta Marte continua hoy en día mediante el rover Curiosity, enviado al planeta rojo en 2012.

Los robots aquí mencionados, son únicamente una porción de los tantos que se han diseñado, sin embargo, es posible notar que las aplicaciones de estos son vastas y que las mismas son ilimitadas debido al desarrollo cada vez más vertiginoso de la tecnología. Los robots móviles brindan la posibilidad de navegar en distintos terrenos y tienen aplicaciones como exploración minera, exploración planetaria, misiones de búsqueda y rescate de personas, limpieza de desechos peligrosos, automatización de procesos, vigilancia, reconocimiento de terreno, y también son utilizados como plataformas móviles que incorporan un brazo manipulador.

Los robots móviles se pueden clasificar por el tipo de locomoción utilizado, en general, los tres medios de movimiento son por ruedas, por patas y orugas. Cabe señalar que aunque la locomoción por patas y orugas han sido ampliamente estudiadas, el mayor desarrollo se presenta en los robots móviles con ruedas (RMR). Dentro de los atributos más relevantes de los RMR, destacan su

eficiencia en cuanto a energía en superficies lisas y firmes, a la vez que no causan desgaste en la superficie donde se mueven y requieren un número menor de partes y menos complejas, en comparación con los robots de patas y de orugas, lo que permite que su construcción sea más sencilla. Son precisamente estos argumentos los que motivan el análisis de este tipo de robots. De esta manera, se puede definir un robot móvil de ruedas como un sistema electromecánico controlado, que utiliza como locomoción ruedas de algún tipo, y que es capaz de trasladarse de forma autónoma a una meta preestablecida en un determinado espacio de trabajo. Se entiende como autonomía de un robot móvil, al dominio que tiene éste para determinar el curso de su acción, mediante un proceso propio de razonamiento en base a sensores, en lugar de seguir una secuencia fija de instrucciones.

Existen diferentes configuraciones cinemáticas para los RMR. Estas dependen principalmente de la aplicación hacia donde va enfocado, no obstante, de manera general se tienen las siguientes configuraciones:

- Ackerman.
- Triciclo clásico.
- Tracción diferencial.
- Skid steer.
- Síncrona.
- Tracción omnidireccional.

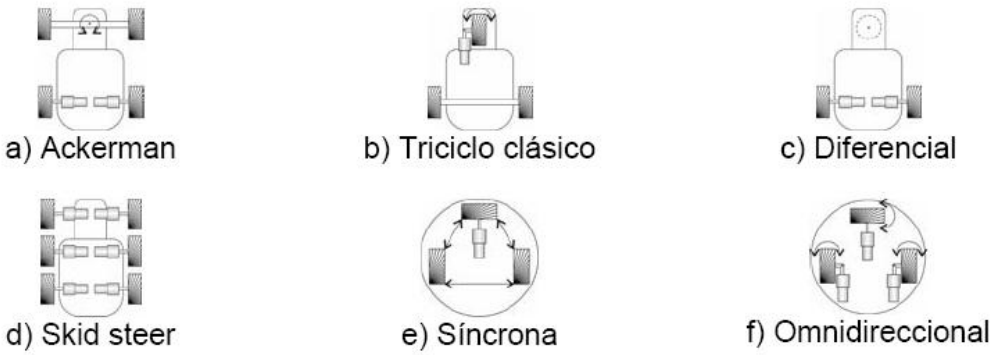

Figura 1.1: Configuraciones cinemáticas más habituales en robots que usan ruedas.

Dependiendo de la configuración cinemática que lo conforme, los RMR utilizan cuatro tipos de ruedas para su locomoción:

- Convencionales.
- Tipo castor.
- Ruedas de bolas
- Omnidireccionales.

Figura 1.2: Tipos de ruedas.

En el marco de las configuraciones cinemáticas posibles y las ruedas que éstas utilizan, los RMR documentados en la literatura utilizan comúnmente la configuración de tracción diferencial donde se utilizan ruedas convencionales como ruedas motrices y una o dos ruedas tipo castor, de bola, u omnidireccionales respectivamente, para proveer de estabilidad al móvil. Esta configuración es sencilla de construir y de controlar, de ahí su gran difusión, ya que es suficiente un par de actuadores para lograr movimiento. Además para conseguir el movimiento de rotación tan sólo hay que invertir el sentido de giro de las ruedas motrices y por lo tanto el control de ese movimiento es bastante sencillo.

Con el objeto de hacer más tratable el problema del modelado en las configuraciones cinemáticas, se suelen establecer algunas suposiciones de diseño y de operación. Por una parte, dentro de las suposiciones de diseño generalmente se toman tres. La primera va dirigida a considerar que las partes dinámicas del RMR son insignificantes, es decir, que no contiene partes flexibles, de esta manera pueden aplicarse mecanismos de cuerpo rígido para el modelado cinemático. La segunda limita que la rueda tenga a lo más un eslabón de dirección, con la finalidad de reducir la complejidad del modelado. La tercera es asumir que todos los ejes de dirección son perpendiculares a la superficie, de esta manera se reducen todos los movimientos a un solo plano.

Por otra parte, respecto a las suposiciones de operación, al igual que en las de diseño, se toman tres. Una de ellas descarta toda irregularidad de la superficie donde se mueve el RMR. Otra, considera que la fricción de traslación en el punto de contacto de la rueda con la superficie donde se mueve, es lo suficientemente grande para que no exista un desplazamiento de traslación del móvil. Como complemento a lo anterior, una tercer suposición de operación establece que la fricción rotacional en el punto de contacto de la rueda con la superficie donde se mueve, es lo suficientemente pequeña para que exista un desplazamiento rotatorio. Aunque las suposiciones mencionadas son realistas, el deslizamiento que ocurre en el punto de contacto de las ruedas con la superficie se ha convertido en un tópico importante debido a las repercusiones que tiene sobre el móvil.

En el ámbito de los actuadores utilizados para dotar de movimiento a los RMR es común que se utilicen motores. Existe una gama bastante amplia dependiendo de su empleo. Los más utilizados en la robótica móvil son los de corriente directa (DC), debido a que su modelo es lineal, lo que facilita enormemente su control. Los motores de DC de imán permanente se pueden clasificar en dos categorías, con escobillas y sin escobillas. Ambos tipos brindan ventajas semejantes, sin embargo, los motores sin escobillas tienen algunas ventajas significativas sobre los motores con escobillas, como por ejemplo:

- Al no contar con escobillas, no se requiere el reemplazo de éstas ni mantenimiento por residuos originados de las mismas.
- No se generan chispas por el contacto de las escobillas. De esta forma se pueden considerar más seguros en ambientes con vapores o líquidos inflamables.
- La interferencia causada por la conmutación mecánica de las escobillas se minimiza considerablemente mediante una conmutación electrónica.
- Los motores sin escobillas alcanzan fácilmente velocidades de hasta 50000 rpm. En cambio los motores con escobillas apenas pueden llegar a las 5000 rpm.

Actualmente el control de los RMR acapara la atención de gran cantidad de investigadores. Desde el punto de vista de la teoría de control, estos se encuentran en el área que se conoce como control de sistemas no-holónomos. Los sistemas no-holónomos se caracterizan por tener un número de grados de libertad controlables menor que el número de grados de libertad totales. En el caso de un RMR de tracción diferencial, el número total de grados de libertad es 3, posición, es

decir, coordenada x e y, y orientación, coordenada θ. Sin embargo únicamente se puede controlar el desplazamiento hacia adelante y hacia atrás y la orientación, quedando como incontrolable el desplazamiento transversal. Matemáticamente se dice que el sistema esta sujeto a restricciones no integrables en las velocidades, es decir, su plano de velocidades está restringido.

El control del movimiento de los RMR, a grandes rasgos, se puede clasificar en cuatro tareas fundamentales:

- Localización.
- Planificación de trayectorias.
- Seguimiento de trayectorias.
- Evasión de obstáculos.

Uno de los métodos más empleados en el seguimiento de trayectoria es el de persecución pura (pure pursuit en inglés). Éste genera arcos entre las ubicaciones del robot móvil y los puntos de la trayectoria a seguir. Los arcos se generan de 5 a 15 veces por segundo lo que resulta en un seguimiento suave y con muy buenos resultados. Precisamente este área de trabajo es el que ha motivado el desarrollo de este proyecto. En los sucesivos capítulo del presente documento de describirán en detalle todas las actividades llevadas a cabo durante la realización del proyecto.

Capítulo 2

Plataforma robótica Manfred

El robot Manfred (MAN FRiEnDly) es un manipulador móvil de 8 grados de libertad construido por el grupo de investigación Robotics Lab perteneciente al departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid. El robot Manfred se usa para probar de forma experimental los algoritmos desarrollados en el grupo de investigación, focalizados en las áreas de modelado 2D y 3D, localización global 2D y 3D, aprendizaje, manipulación de objetos, planificación de trayectorias, formaciones de robots, etc. Estas habilidades permiten a Manfred desenvolverse de forma autónoma en entornos adaptados a seres humanos, evitando los obstáculos que encuentre a su paso y realizando con mucha eficacia diversas de tareas de manipulación de objetos.

La apariencia de Manfred está inspirada en los vehículos de exploración planetaria (rovers); consta de una base móvil diferencial (2 grados de libertad) y de un brazo antropomórfico ligero de 6 grados de libertad capaz de realizar las tareas que un brazo humano pueden hacer, por ejemplo, abrir y cerrar puertas, agarrar y manipular objetos, trasladarlos de un lugar a otro, etc. Manfred está constituido por numerosos sistemas independientes (sistema sensorial, sistema locomotor, etc.) que se conectan para conseguir un funcionamiento adecuado de la plataforma. El diseño modular del robot, con interfaces bien definidas a nivel hardware (interfaces mecánicas y eléctricas) y software (interfaces de comunicación entre programas) simplifica la integración de todos esos sistemas y agiliza su mantenimiento y futura ampliación.

La descripción de Manfred que se llevará a cabo en el resto del capítulo tratará acerca de:

- Estructura mecánica y sistema eléctrico.

- Sistema de manipulación: brazo manipulador LWR-UC3M-1 y sensor de fuerza/par JR3.
- Sistema de percepción del entorno: telemetría láser y cámara 3D.
- Sistema de control: tarjeta controladora de ejes PMAC2-PCI.
- Sistema de procesamiento central: ROS.

Figura 2.1: Robot manipulador Manfred.

2.1. Estructura mecánica y sistema eléctrico

En la estructura que constituye el robot Manfred se aprecian dos partes bien diferenciadas; una base octogonal en la que se ubica el sistema de alimentación del robot y sobre ella el mástil al que se une el brazo manipulador. La base, con una altura de 235 mm, está constituida por dos planchas de aluminio con forma de octógono irregular dispuestas una encima de la otra.

La base cuenta con la anchura suficiente para albergar en su interior 4 baterías y la electrónica de potencia de los motores de las ruedas, pero lo suficientemente "estrecha" para circular por pasillos, corredores y puertas de dimensiones "normales"[1] sin hacer maniobras complicadas. El

[1] Por "normal" se quiere dar a entender las dimensiones que por término medio tienen los entornos adaptados a los seres humanos.

conjunto de baterías alojado en el interior de la base móvil actúa de contrapeso, equilibrando las fuerzas y momentos aplicados en el robot aún cuando el brazo manipulador realice tareas en posiciones críticas, dando estabilidad al conjunto. El robot Manfred puede extraer la energía necesaria para su funcionamiento de dos fuentes diferentes, bien del conjunto de 4 baterías de 12 volt y 45 A/h, conectadas en serie para proporcionar 48 V, o bien de la red eléctrica. Los sensores y dispositivos electrónicos que usa Manfred pueden necesitar distintas tensiones de alimentación, por lo que es necesario disponer de distintos conversores DC/DC. En la parte posterior de la base móvil existe un panel constituido por tres interruptores, dependiendo de su configuración Manfred extrae y emplea la tensión de alimentación de un modo diferente.

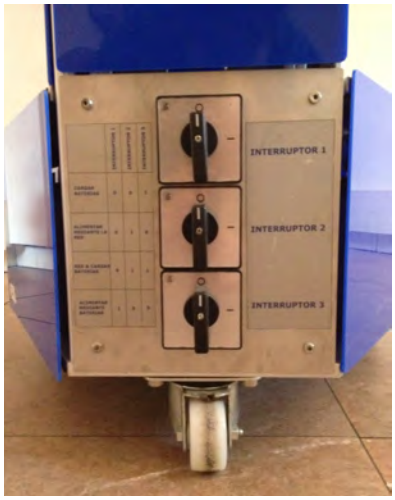

Figura 2.2: Interruptores que seleccionan el modo de alimentación de Manfred.

1. **INT1 = OFF, INT2 = OFF, INT3 = ON:** Recargar baterías. Este es el modo de alimentación que se usa cuando sólo se quiere recargar las baterías del robot, después de una fase de prueba de algoritmos, mientras el resto del robot permanece apagado.

2. **INT1 = OFF, INT2 = ON, INT3 = OFF:** Alimentación mediante la red eléctrica. Este es el modo de alimentación utilizado cuando se están haciendo pruebas de código, depuración de código,pruebas menores, etc. Es el modo de trabajo habitual durante la fase de desarrollo de los algoritmos que posteriormente usará el robot.

3. **INT1 = OFF, INT2 = ON, INT3 = ON:** Alimentar mediante la red eléctrica y al mismo tiempo recargar las baterías. Este modo de alimentación permite recargar las

baterías del robot después de una fase de prueba de algoritmos mientras se sigue trabajando en el desarrollo y mejora de estos.

4. **INT1 = ON, INT2 = OFF, INT3 = OFF:** Alimentación mediante baterías. Todos los sistemas que forman parte del robot se alimentan de las baterías ubicadas en la base. Este es el modo de alimentación seleccionado cuando se quiere dotar de autonomía al robot y probar los algoritmos desarrollados de cualquier clase, por ejemplo, navegación, localización, manipulación, visión, etc.

La base cuenta con dos mecanismos de regulación manual de la altura de las ruedas motrices, uno por rueda, permitiendo cambiar la distancia a la que se encuentra la base del suelo. Usando este sistema el robot puede acercarse al suelo algo más de 80 mm, aproximando su centro de gravedad al mismo, mejorando su estabilidad. Cada sistema de regulación está constituido por dos bloques de aluminio, dispuestos uno encima del otro. El bloque superior se haya fijado a la plancha superior de la base mientras que el bloque inferior está unido a una de las ruedas y al bloque superior por medio de varios tornillos pasantes. Ajustando la posición de los tornillos que unen ambos bloques se consigue modificar la altura del robot al suelo.

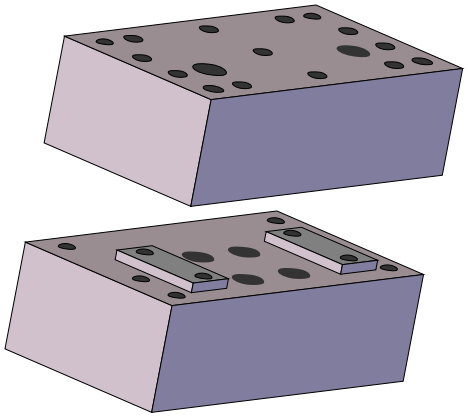

Figura 2.3: Mecanismo de regulación manual de la posición de una rueda motriz.

Las dos ruedas motrices que usa Manfred son de la casa Rockland Bayside, modelo DX 6-20-1M-P Servo-Wheels, cada una de las cuales incorpora un motor sin escobillas (brushless en inglés) y una reductora de tipo 20:1.

Figura 2.4: Imagen del mecanismo de regulación manual de la altura de una rueda motriz en Manfred.

Para conocer el desplazamiento de cada rueda y utilizar esta información para averiguar la ubicación del robot en cualquier instante de tiempo, se han montado junto a éstas unos encoder ópticos relativos de alta resolución, 512 cts/rev, de la casa HP modelo HEDS-5540-A06.

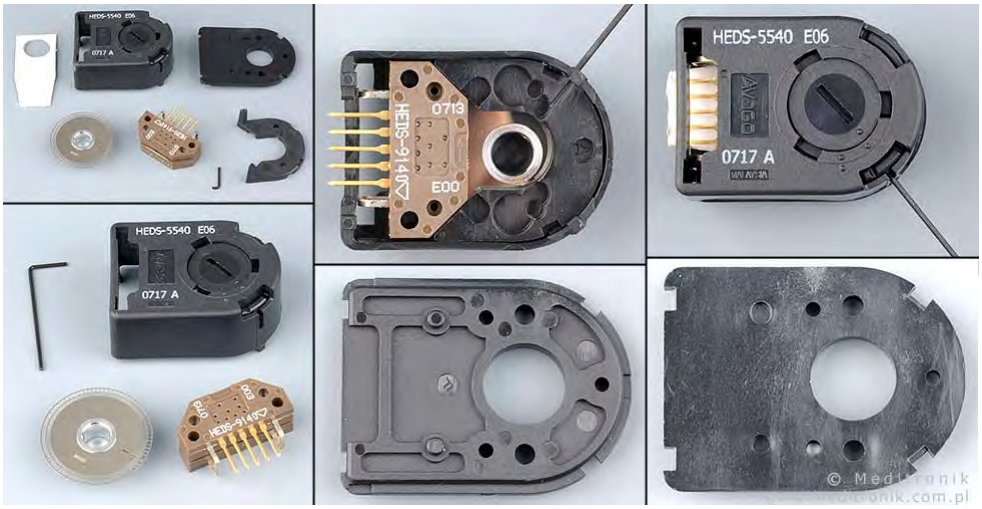

Figura 2.5: Encoders usados en los motores de las ruedas motrices de Manfred.

Para que la base sea estable se ubican tres ruedas pasivas próximas a las ruedas motrices, dos de las cuales se hayan en la parte delantera de la base y la restante en la parte trasera. El mástil del robot, con una altura cercana a los 105 cm, está fijado por múltiples lugares tanto a la plancha superior como inferior de la base, lo que aumenta la rigidez del conjunto.

El mástil sirve de soporte en el que fijar el brazo manipulador de 6 grados de libertad con

el que actualmente cuenta el robot. La disposición del brazo manipulador en el mástil es tal que cuando éste se haya en su posición de reposo está considerablemente alejado del resto de elementos que conforman el robot, de manera que si durante una tarea el sistema se queda sin tensión, el brazo regresa a su posición de reposo[2] sin chocar contra otros elementos, evitando su daño y el del resto de elementos.

El mástil cuenta con otra zona de anclaje en la que disponer un segundo brazo manipulador, en desarrollo actualmente, confiriendo al robot una apariencia antropomórfica. Además de actuar como soporte de anclaje el mástil se usa para albergar en su interior todo tipo de elementos.

Figura 2.6: Componentes que guarda Manfred en su interior.

A Conversor 48 V DC a 24 V DC para alimentar diferentes dispositivos del robot, por ejemplo el láser SICK S3000.

B Conversor 48 V DC a 12 V para alimentar diferentes dispositivos del robot, por ejemplo la cámara Kinect de Microsoft.

C Conversor traco-power de 48 V DC a ±15 V DC para alimentar unos dispositivos electrónicos (de color verde en la figura 2.6), denominados accesorios 8E, cuya función se detalla más adelante, al hablar de la tarjeta controladora de ejes que incorpora el ordenador central del robot.

[2]Los motores del brazo carecen de frenos para no aumentar en exceso su peso.

D Caja de fusibles para proteger diferentes elementos del robot, como por ejemplo los conversores DC-DC, el multímetro, los accesorios 8E, el láser SICK S3000, etc.

E Fuente de alimentación del ordenador central de Manfred.

F Drivers de potencia de los motores del brazo manipulador. Los drivers de los motores del brazo se ubican en el soporte vertical mientras que los drivers de los motores de las ruedas se hayan en la base.

G Fusibles para proteger los drivers de los motores del brazo.

H Interfaces de conexión de los cables que provienen del brazo (cables para la fase de los motores, cables de los sensores de efecto hall de los motores, cables de los sensores inductivos usados para buscar la posición de reposo del brazo, cable RJ45 del sensor de fuerza/par ubicado cerca del extremo final del brazo, etc.) y que se dirigen hacia los drivers y a los accesorios 8E.

Hay que destacar que el mástil se diseñó pensando en la posibilidad de incluir en el futuro un segundo brazo manipulador, por lo que cuenta con espacio libre para albergar los elementos que use el nuevo brazo como se puede observar en la figura 2.6. El mástil cuenta con una consola que centraliza el acceso a las partes mas relevantes del robot:

- Botón de encendido del ordenador que gobierna el comportamiento del robot (botón plateado) y botón de reset (botón negro).
- Conector RS-232 de comunicación serie(en azul).
- Conector VGA para conectar un monitor a Manfred y poder interaccionar con el software instalado en el robot.
- Un hub usb de 4 tomas para conectar numerosos dispositivos de este tipo, por ejemplo, teclado, ratón, pen-drive usb, etc.
- Toma de red ethernet para conectar el robot a internet. Hay que destacar que el robot posee hardware de conexión inalámbrica wifi, por lo que la toma de ethernet sólo se usa en ocasiones excepcionales cuando se quiere descargar algún contenido digital al robot con una velocidad mayor que la que se obtiene con la conexión inalámbrica.

Figura 2.7: Consola central de Manfred.

- Cartel que nos recuerda la configuración que deben tener los conmutadores para seleccionar los diferentes modos de alimentación del robot, y el umbral de descarga de las baterías que no se debe exceder para un uso correcto de las mismas.

- Interruptor para encender/apagar los sensores/actuadores disponibles en Manfred, láser, cámaras (de todo tipo), pinza del extremo del brazo, etc. Existen varios interruptores sin usar a la espera de que en un futuro se puedan añadir más elementos que puedan necesitarlos.

- Panel LCD de un multímetro que nos indica, activándolo por medio de un interruptor de los citados en el punto anterior, cual es la tensión de las baterías (en su conjunto, no individualmente).

- Conmutador que permite alimentar los motores del robot. Se puede trabajar con el ordenador del robot sin necesidad de tener los motores encendidos, lo que supone un ahorro considerable de energía.

- Seta de seguridad. Si ocurriese alguna situación de emergencia que exigiera la parada inmediata del robot, la pulsación de esta seta corta el suministro eléctrico de todo el sistema.

Para proteger la electrónica que incorpora el robot en su interior y para dotar al robot de una apariencia más elegante la mayor parte de la estructura ha sido recubierta por piezas de policarbonato del color corporativo de la Universidad Carlos III de Madrid que se adhieren a ella mediante pegatinas imantadas. Este mecanismo de fijación permite un rápido y fácil acceso al interior del robot cuando sea necesario.

2.2. Sistema de manipulación: brazo LWR-UC3M-1 y sensor de fuerza/par JR3.

El robot Manfred posee un brazo manipulador con 6 grados de libertad, desarrollado en la Universidad Carlos III de Madrid por el grupo de investigación Robotics Lab, denominado LWR-UC3M-1 (LightWeight aRm UC3M 1, con el que puede realizar numerosas tareas de manipulación y colaboración con personas. Los grados de libertad del brazo LWR-UC3M-1 están distribuidos del siguiente modo:

- Hombro: 2 g.d.l que posibilitan giros en el eje anteroposterior y transversal.
- Codo: 1 g.d.l que posibilita un movimiento similar al humano.
- Muñeca:3 g.d.l; uno asimilable a la pronosupinación; otro comparable con la aducción-abducción y flexión-extensión según la posición relativa con el anterior y el último ofrece también rotación en la pronosupinación de manera que se obtiene redundancia en la orientación.

Es obvio que este modelo cinemático no tiene una correspondencia exacta con la del modelo humano, sin embargo, la falta de g.d.l se resuelve de la siguiente manera:

1. En el caso del hombro humano, en el cual existe un tercer g.d.l, éste se suple con movimientos de giro y rotación de la plataforma móvil.
2. En el caso de la muñeca, para la mayoría de movimientos humanos no se combinan los g.d.l referentes a la flexión-extensión y a la aducción-abducción, por lo que esta distribución de g.d.l no desvirtúa el diseño antropomórfico.

Figura 2.8: Brazo antropomórfico ligero de 6 grados de libertad LWR-UC3M-1.

Uno de los objetivos principales durante el diseño del brazo manipulador LWR-UC3M-1 ha sido conseguir que su peso no fuera excesivo, manteniendo a la vez rigidez y resistencia en la estructura. Para cumplir con este requisito se ha llevado a cabo un análisis exhaustivo de las propiedades de aquellos materiales que pudieran ser usados para construir el brazo. Entre las características estudiadas de los materiales candidatos destaca su capacidad de ser mecanizado, dada la necesidad de fabricar diferentes piezas con formas, en muchos casos, complejas. Se han analizado varios materiales compuestos tanto metálicos como polímeros.

Esta característica no está presente en el resto de los manipuladores ligeros desarrollados por otros grupos de investigación. Entre los compuestos metálicos se estudiaron aleaciones de magnesio (desechadas debido a la dificultad de su mecanizado), aleaciones de litio (presentan como principal inconveniente la peligrosidad del contacto directo con estos materiales), aleaciones de titanio (tienen una densidad elevada y un elevado coste para nuestro objetivo) y, por último, las aleaciones de aluminio, que ha sido una de las opciones seleccionadas. La aleación de aluminio seleccionada (7075), aunque presenta una densidad superior a la de otros compuestos metálicos analizados, $2'8\,\mathrm{g/cm^3}$, no presenta sus inconvenientes. Es un material de fácil mecanizado, sin inconvenientes en cuanto a su peligrosidad o toxicidad. Por último, se consideró también la utilización de fibra de carbono, de densidad $1'6\,\mathrm{g/cm^3}$. La mayoría de las partes estructurales se han fabricado con aluminio, mientras que los eslabones 3, 4, 5 y 6 se han construido con fibra de carbono.

Otro de los objetivos de diseño del brazo manipulador LWR-UC3M-1 es evitar que alguno de sus elementos pueda dañar a un ser humano. Para cumplir con este objetivo se ha puesto

especial atención en evitar que la estructura mecánica externa del manipulador posea aristas. Para ello se han redondeado los extremos de los elementos que cubren las articulaciones y se han ocultado los tornillos en la propia estructura mecánica. Al presentar una estructura cerrada y tubular con un hueco en su interior de diámetro adecuado, los eslabones de fibra de carbono pueden contener electrónica, cables dedicados a control y alimentación, etc., de manera segura y limpia. Esta estructura cerrada permite reducir de una manera significativa los peligros que pudieran ocasionar las vibraciones e inexactitudes de posición en la estructura, dándole una robustez considerable al diseño.

Cada una de las articulaciones esta compuesta por un motor sin escobillas de corriente continua, modelo RBE de la firma Kollmorgen, un equipo de reducción de velocidad y aumento de par del tipo Harmonic Drive, modelo HFUC-2UH, un encoder óptico relativo de alta resolución (1024 cts/rev) de la casa HP, modelo HEDS550, y un sensor inductivo de presencia usado para obtener la posición de reposo de cada articulación. Tanto los motores de la base como los del brazo son sin escobillas, por tanto no producen chispas, lo que permite que el robot sea usado en ambientes en los que haya gases explosivos u otros materiales peligrosos.

Una de las características más notables del brazo LWR-UC3M-1 es la ausencia de frenos. Esta característica no está presente en el resto de los manipuladores ligeros desarrollados por otros grupos de investigación. Los reductores incorporados en cada articulación junto con la disposición antropomórfica y las configuraciones previstas para el manipulador en el desarrollo de sus tareas, permiten prescindir de la incorporación de sistemas de frenado en las articulaciones con la consiguiente reducción de coste y peso global.

Con estas pautas de construcción la masa del conjunto del brazo es de unos 18 kg, con una capacidad de carga máxima en el extremo de unos $4'5$ kg, lo que resulta en una relación carga/peso entre 1:3 y 1:4, y tiene un alcance máximo de 955 mm. El brazo LWR-UC3M-1 se haya montado en el lateral derecho del mástil de Manfred, de modo que tanto el sistema de visión como la telemetría láser 3D pueden percibir los objetos que se desean manipular sin ser obstaculizados por el brazo.

G.D.L	Recorrido [°]	ω [°/s]	Cuentas en encoder
1, 2, 3	160	131	1024
4, 5	120	180	1024
6	100	210	1024

Tabla 2.1: Características del brazo manipulador LWR-UC3M-1.

Para que Manfred realice tareas de manipulación de manera segura y eficaz (agarrar el pomo de una puerta, presionar un interruptor, etc.) posee en el extremo de su brazo un sensor de fuerza/par de la casa JR3, modelo 67M25A50I40, que cuenta con una capacidad de carga de 11 kg, un peso de $0'175$ kg y una frecuencia de adquisición de muestras de 8 kHz.

Figura 2.9: Sensor de fuerza/par JR3.

El sensor JR3 posee varios sistemas de galgas extensiométricas que generan 6 señales analógicas proporcionales a la fuerza y al par aplicado en los ejes de coordenadas X, Y, Z del sensor. El eje X y el eje Y se hayan en el plano transversal del sensor mientras que el eje Z coincide con su eje central. El punto de referencia para todos los datos de carga es el centro geométrico del sensor. El sensor JR3 que emplea Manfred se sitúa entre el extremo del brazo, justo antes de una pinza usada para manipular objetos.

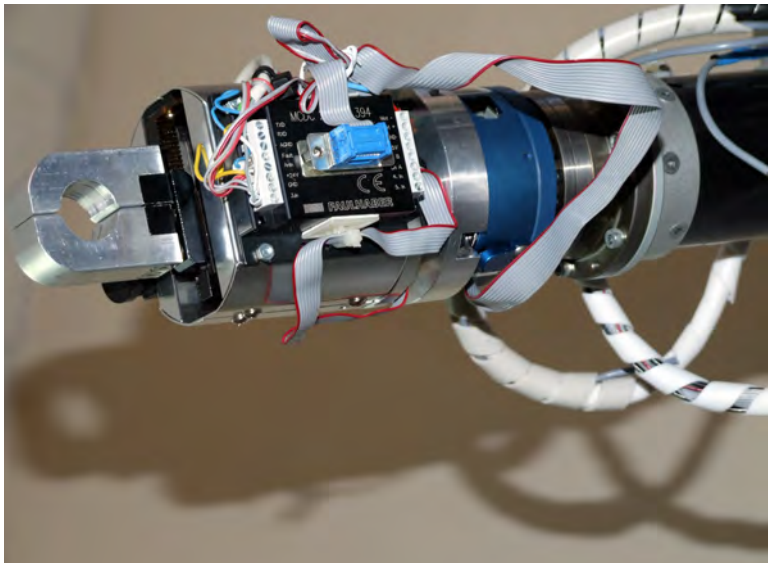

Figura 2.10: El sensor JR3 que utiliza Manfred se ubica en el extremo del brazo, antes de la pinza.

El sensor JR3 está acompañado de un sistema de procesamiento digital de señales o DSP, modelo ADSP-2184 de la casa Analog Devices, que amplifica, filtra y digitaliza las señales analógicas proporcionadas por el sensor de fuerza/par, obteniendo una buena relación señal a ruido. El sensor de fuerza/par está conectado a través de una cable RJ45 a la tarjeta PCI que contiene el DSP y a su vez ésta se haya conectada al ordenador central de Manfred.

Esta tarjeta receptora de datos se usa para monitorizar y ajustar la tensión de alimentación del sensor JR3, filtrar datos paso bajo (permitiendo diferentes frecuencias de corte), realizar operaciones de rotación y transformación de coordenadas, cálculo de magnitudes vectoriales, captura de picos máximos y mínimos, etc. La tarjeta PCI se comunica con el computador de Manfred a través de una zona de memoria compartida ubicada en el espacio de direcciones que el sistema operativo de Manfred (Ubuntu 11.10) destinada al bus PCI.

Figura 2.11: Tarjeta PCI que incorpora el DSP ADSP-2184 de la casa Analog Devices.

2.3. Sistema de percepción del entorno: telemetría láser y cámara 3D

Un robot completamente autónomo debe extraer información del entorno en el que se haya e interpretarla adecuadamente para poder ejecutar diversas tareas. La función del sistema de telemetría láser es obtener la distancia que separa al robot de los objetos que hay a su alrededor. Esta información la puede usar el robot para construir un modelo del espacio de trabajo donde se haya, para localizarse en dicho entorno, para navegar entre dos puntos del entorno, etc. Según la complejidad y el grado de ocupación del entorno de trabajo se utiliza telemetría 2D o 3D.

Para conseguir estos objetivos Manfred cuenta con varios sensores de percepción del entorno, un telémetro láser de la casa SICK, modelo S3000 Professional CMS, un telémetro láser de la casa Hokuyo, modelo UTM-30LX, y una cámara 3D de la casa Microsoft, modelo Kinect.

Durante el desarrollo de este proyecto se ha usado únicamente el telémetro láser SICK S3000, por ello se describirán exhaustivamente las características de dicho telémetro láser, su principio físico de funcionamiento (común con el Hokuyo UTM-30LX), y se detallará como ha sido usado en la práctica. A continuación se hará una breve descripción de las características del telémetro láser Hokuyo UTM-30LX, y finalmente se describirá brevemente la cámara 3D Kinect y su principio físico de funcionamiento.

2.3.1. Telémetro láser SICK S3000

Manfred lleva instalado sobre su base el telémetro láser bidimensional SICK S3000 Professional CMS (S3000 PCMS) capaz de detectar los obstáculos que se encuentren a 35 cm sobre el suelo y cuya distancia esté comprendida entre 0′1 m y 10 m; por ejemplo mobiliario de interiores, piernas de personas, etc.

Figura 2.12: Telémetro láser bidimensional SICK S3000 Professional CMS.

Este láser ha sido construido para ser usado en entornos industriales, en donde las condiciones de seguridad y fiabilidad son muy exigentes, por ello incorpora numerosas opciones de configuración (alertas, parada de emergencia controlada, numerosos campos advertencia configurables por distancia, trabajo colaborativo con otros láseres y/o dispositivos de control en una red con arquitectura de bus, por ejemplo PROFINET, etc) que son de gran ayuda.

El funcionamiento de la telemetría láser consiste en emitir haces de luz, normalmente infrarroja y de corta duración (S) con diferentes orientaciones. Las diferentes orientaciones con las que son emitidos los haces de luz se consiguen con un espejo que rota sobre sí mismo con una velocidad uniformemente. En el caso del telémetro láser S3000 PCMS el ángulo abarcado es de 190° (1′055 · π rad).

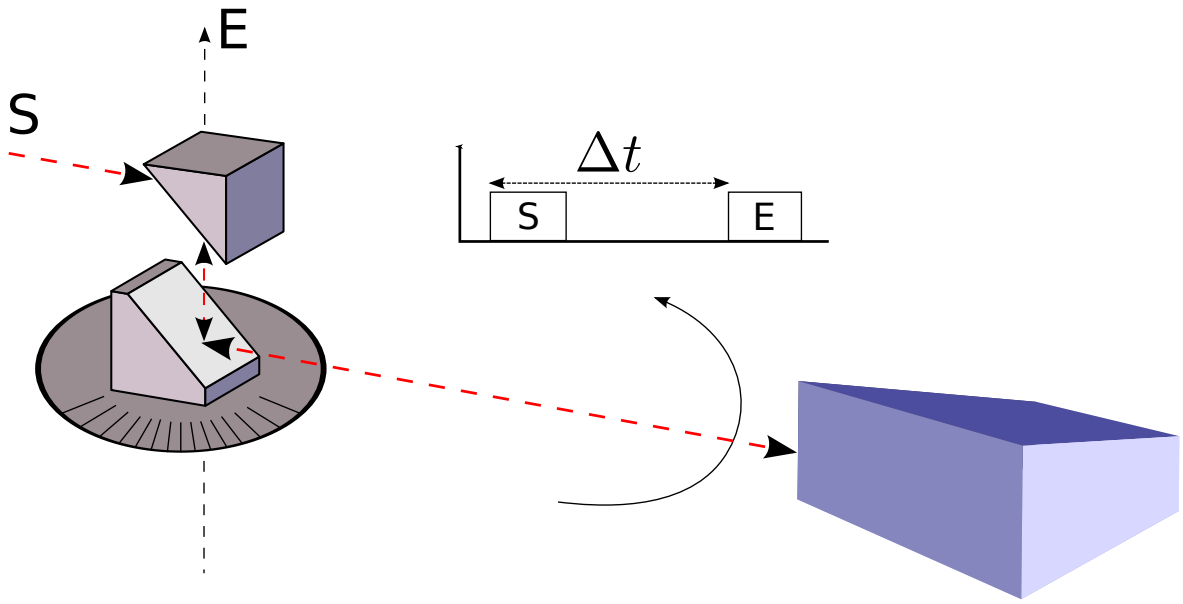

Figura 2.13: Principio de funcionamiento de la telemetría láser.

Simultáneamente a la emisión de cada haz se pone en marcha un cronómetro electrónico. Cada haz se aleja del telémetro a la misma velocidad constante; la velocidad de la luz. Cuando un haz de luz de los emitidos incide en un obstáculo es reflejado y regresa al emisor. Con la llegada de cada haz al telémetro se para el cronómetro correspondiente.

Teniendo en cuenta el intervalo de tiempo transcurrido entre el instante de emisión y recepción de cada haz de luz, y la velocidad de propagación constante de estos, el S3000 calcula la distancia que le separa de los obstáculos según la expresión 2.1, garantizando un error máximo de 30 mm mm en las medidas.

$$d_{obs} = \frac{c \cdot t_{prop}}{2} \tag{2.1}$$

El S3000 no necesita receptores externos ni reflectores para funcionar, proporcionando las siguientes ventajas:

1. La instalación requiere menores costes y despliegue de medios.
2. La zona supervisada puede configurarse a medida del usuario.
3. En comparación con los sensores táctiles, la exploración sin contacto conlleva un menor desgaste.

El láser SICK S3000 permite configurar por software hasta 8 campos de protección que son usados para evitar que el robot acceda a zonas peligrosas. En el robot Manfred se han configurados sólo dos campos de protección. El primero de ellos, denominado *campo de protección de advertencia*, abarca un ángulo de 190° y 50 cm de radio alrededor del láser. El segundo campo de protección, denominado *campo de protección de peligro*, abarca un ángulo de 190° y 30 cm de radio alrededor del láser. En cuanto el escáner láser percibe un objeto dentro de un campo protección lo notifica para que el software del robot pueda actuar en consecuencia.

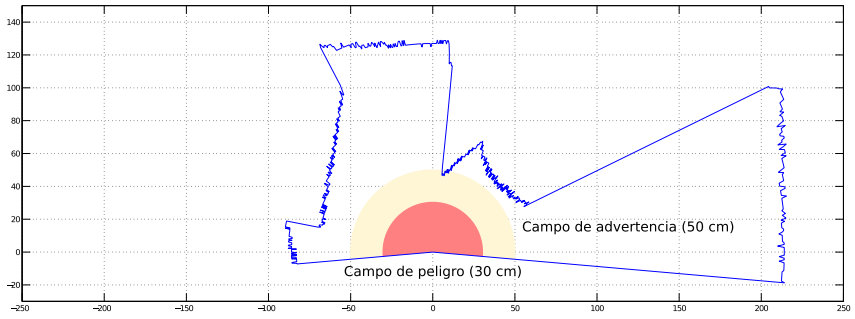

Figura 2.14: Campos de protección del SICK S3000 usados en Manfred.

El láser SICK S3000 puede configurarse para adquirir medidas con una resolución de 0′25° (761 medidas) o 0′50° (381 medidas). La opción con la que actualmente se trabaja consiste en tener el láser configurado para que aporte medidas cada 0′25° y a continuación por software se descartan aquellas medidas que no son útiles. De este modo es sencillo obtener datos cada 0′25°, 0′50° 0′75°, 1°, 1′25°, 1′50°, 1′75°, 2°, etc. Otras características destacables del telémetro láser SICK S3000 son:

- Tiempo de adquisición pequeño, 60 ms cuando se toman 761 medidas.
- Conexión al ordenador central mediante una interfaz serie (RS-422 adaptada a RS-232) a una velocidad máxima de 500 kbps.
- Consumo de 24 V y 800 mA, lo que le hace apto para ser utilizado en sistemas alimentados con baterías como Manfred.

2.3.1.1. Sistema de telemetría 3D

El sistema de telemetría 3D está constituido por un soporte móvil motorizado capaz de girar un ángulo de 120° (90° por encima de la horizontal y 30° por debajo de la horizontal) sobre el que reposa el SICK S3000.

Figura 2.15: Vista frontal del sistema de telemetría láser 3D.

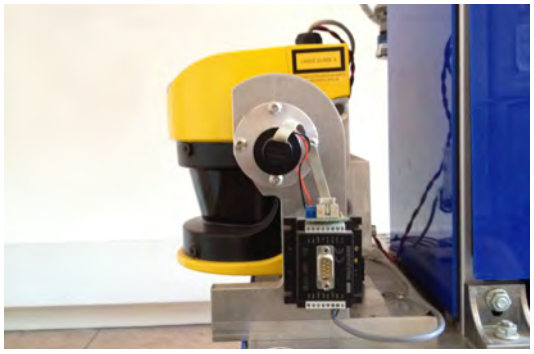

Figura 2.16: Vista lateral del sistema de telemetría láser 3D.

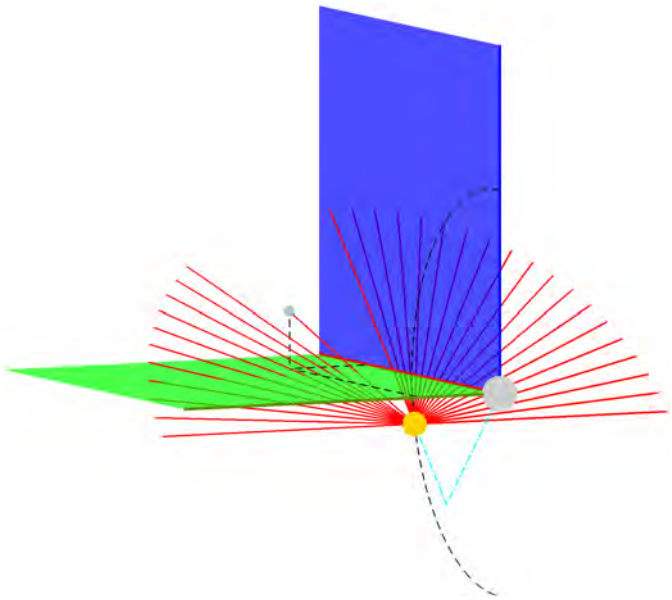

Figura 2.17: Ejemplo de vista 3D básica creada con el sistema de telemetría láser y el soporte móvil. Eje x en rojo, eje y en verde, eje z en azul.

Esta disposición permite al S3000 realizar barridos de diferentes secciones del entorno donde el robot se haya. A continuación estos barridos se combinan para generar un vista 3D. Para rotar el soporte donde se ubica el S3000 se utiliza un motor de continua, de la casa Faulhaber, modelo 2642-024CR, con factor de reducción 1:66, de poco peso y reducidas dimensiones, y el controlador MCDC2805, también de Faulhaber, situado en uno de los laterales del soporte giratorio. Cabe destacar que el controlador MCDC2805 admite control en posición o en velocidad a través de sencillos comandos en formato ASCII.

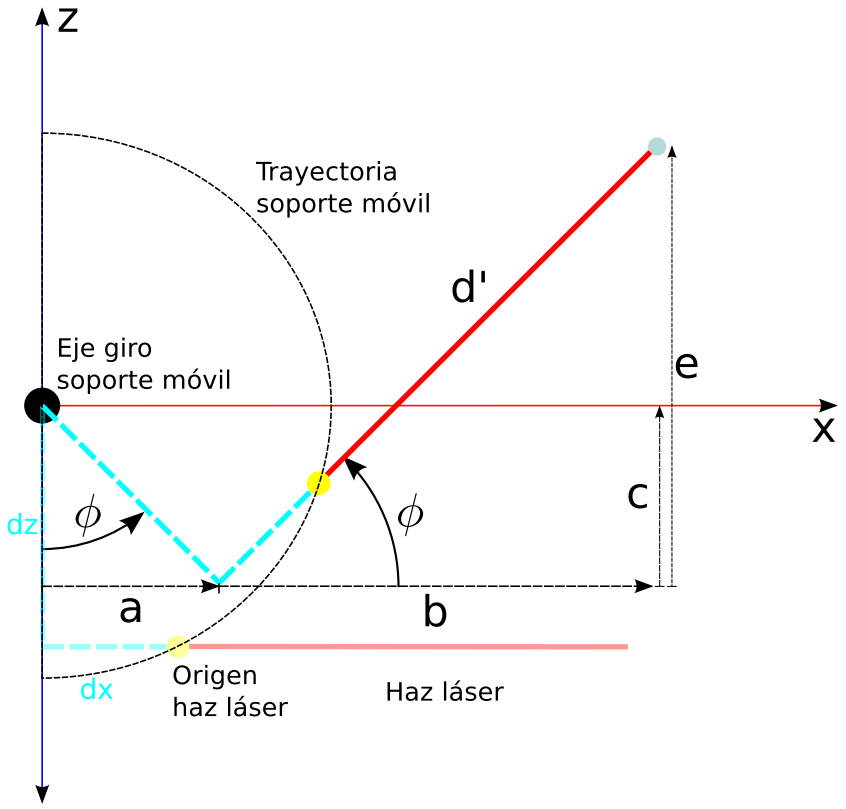

Figura 2.18: Modelo matemático de la telemetría 3D usada en Manfred.

Las coordenadas de un obstáculo encontrado respecto a un sistema de coordenadas ubicado en el punto medio del eje de giro del soporte móvil son:

$$d' = d_{obs} \cdot \cos(\alpha) \tag{2.2}$$

$$x_{obs} = (d' + d_x) \cdot \cos(\phi) + d_z \cdot \sin(\phi) \tag{2.3}$$

$$y_{obs} = d_{obs} \cdot \sin(\alpha) \tag{2.4}$$

$$z_{obs} = (d' + d_x) \cdot \sin(\phi) - d_z \cdot \cos(\phi) \tag{2.5}$$

donde α es el ángulo que forma el haz de luz láser con el plano XZ.

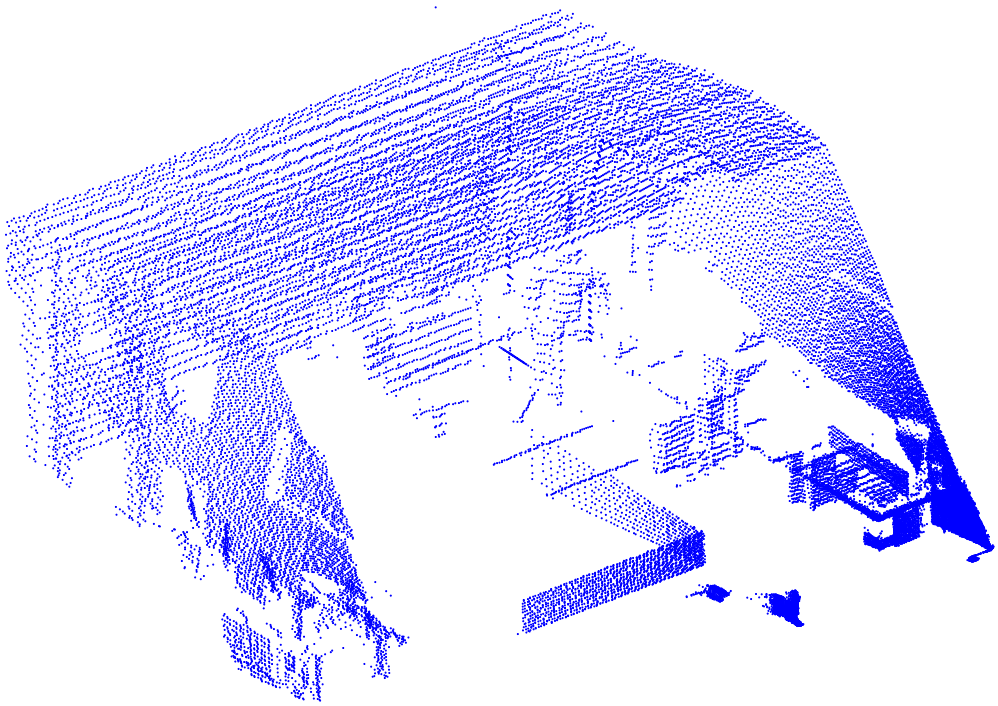

Figura 2.19: Vista del laboratorio de robótica 1.3.C.13 de la Universidad Carlos III creado con el sistema de telemetría 3D de Manfred.

2.3.2. Telémetro HOKUYO UTM-30LX.

El telémetro láser UTM-30LX es el de mejores prestaciones que oferta la empresa Hokuyo. Debido a su elevada apertura angular, 270°, y a su amplio rango de medida, entre 0′1 m y 30 m, este telémetro láser es ideal para reconocer obstáculos o realizar modelados del entorno, sean 2D o 3D, por lo cual se haya situado en la parte frontal del robot. Las características más notables de este sensor son:

- Resolución angular de 0′25°, lo que produce 1081 medidas en la apertura angular completa de 270°.
- Tiempo de adquisición pequeño, 25 ms cuando se toman 1081 medidas.
- Conexión al ordenador central mediante una interfaz USB 2.0 a una velocidad máxima de 500 kbps.
- Consumo de 12 V y 700 mA, lo que le hace apto para ser utilizado en sistemas alimentados con baterías como Manfred.

Figura 2.20: Telémetro láser bidimensional Hokuyo UTM-30LX.

El telémetro Hokuyo UTM 30 LX no ha sido diseñado para operar en entornos industriales, y por tanto no dispone de tantas opciones de configuración como el SICK S3000.

2.3.3. Cámaras 3D Kinect

Kinect es un periférico de la consola Xbox 360 de Microsoft capaz de capturar imágenes en 3 dimensiones. Uno de los sectores más interesados en Kinect es de la robótica ya que una cámara de profundidad asequible abre un mundo de posibilidades para los robots en tareas de navegación y reconocimiento de objetos. La tecnología capaz de lograr estos resultados ya existía desde hace tiempo, pero a un precio muy elevado; sin embargo, Kinect ha supuesto toda una revolución en este ámbito ya que tan sólo cuesta 150 €.

Figura 2.21: Cámara de profundidad Kinect de la compañía Kinect.

Para obtener una imagen 3D de una escena Kinect cuenta con un proyector de luz infrarroja y una sensor que capta sólo este tipo de luz. La intensidad de la luz infrarroja reflejada que llega al receptor indica a que distancia de la cámara se haya ese pixel de la escena. Con esta información se elabora una imagen en escala de grises, de 320 x 240 pixeles, dónde el color más blanco indica más proximidad y el color más oscuro más lejanía. Este proceso se realiza a 30 fotogramas por segundo.

Este método no es el único para construir una imagen 3D de una escena. La visión estéreo puede construir dichas imágenes usando dos o más cámaras, sin embargo, este método es muy dependiente de las condiciones del entorno como la iluminación, etc. El método empleado por Kinect funciona aunque las condiciones de iluminación sean malas, ya que no trabaja en el rango de las frecuencias visibles.

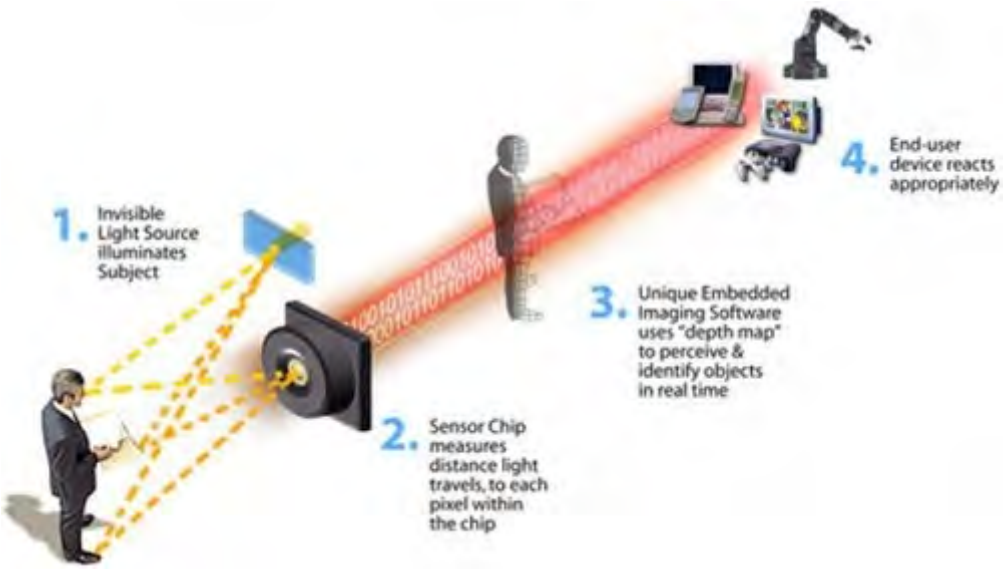

Figura 2.22: Principio físico de funcionamiento de la cámara Kinect.

La información de color y texturas es fundamental, por eso la cámara Kinect cuenta también con una cámara a color (Color MOS) con resolución 640 × 480 pixeles y autofocus capaz de grabar video, hacer reconocimiento facial, etc. Conociendo la distancia que separa ambas cámaras y utilizando diversos algoritmos se puede alinear la imagen de profundidad con la imagen a color. La información resultante de la alineación se puede usar para segmentar los objetos de la escena; es decir, separar un grupo de objetos en sus partes constitutivas, por ejemplo, separar personas del fondo. Adicionalmente la cámara Kinect tiene un motor en la base que permite el movimiento

de la estructura completa para seguir al objetivo.

2.4. Sistema de control: tarjeta controladora de ejes PMAC2-PCI

La tarjeta controladora de múltiples ejes PMAC2-PCI (Programmable Multi Axis Controller) de la compañía Delta Tau Data Systems, Inc. es un dispositivo de alto rendimiento capaz de controlar hasta 8 ejes simultáneamente con un alto nivel de precisión. Gracias a sus más de 1000 variables de configuración y a la capacidad de cómputo de los procesadores digitales de señales actuales (DSP - Digital Signal Processing), la tarjeta PMAC2-PCI puede ofrecer una relación precio-rendimiento muy satisfactoria. El DSP que incorpora la tarjeta PMAC2-PCI es el modelo DSP56002 de 24 bits y frecuencia de trabajo de 40 MHz de la casa Motorola. Existen seis versiones de la tarjeta PMAC2: la PMAC2-PCI, la PMAC2 PC Ultralite, la PMAC2-Lite, la PMAC2 VME, la PMAC2 VME Ultralite y la PMAC2 Mini.

Estas tarjetas se diferencian entre sí en su forma, la naturaleza de los buses de comunicación y de los puertos de E/S que usan. Todas las versiones de la tarjeta tienen prácticamente el mismo firmware, lo que permite que los programas escritos para una versión de la PMAC2 puedan ser ejecutados en otra de las variantes de dicha tarjeta. Cualquier versión de PMAC2 puede funcionar como un controlador independiente o puede ser comandado por un computador, ya sea a través de un puerto serie o mediante el bus PCI.

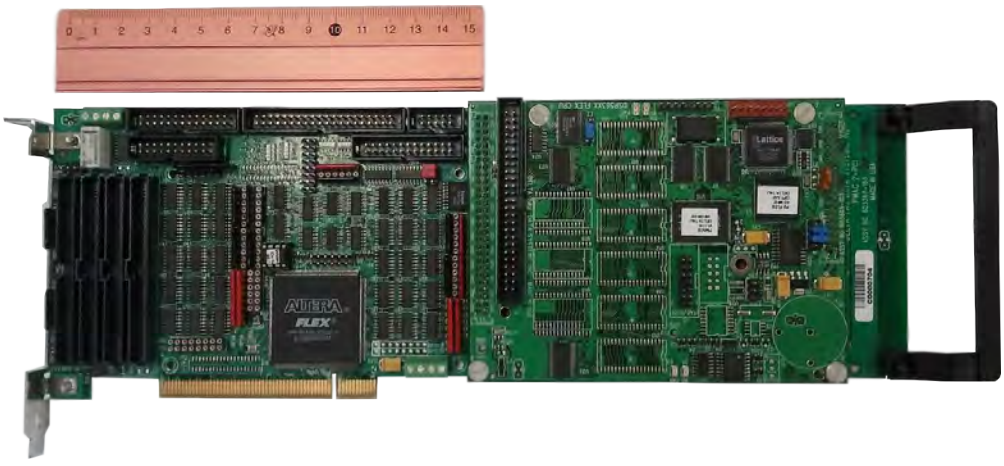

Figura 2.23: Tarjeta controladora de 8 ejes PMAC2-PCI.

Concebida para ser una tarjeta controladora de propósito general, la PMAC2 puede ser usada

en una amplia variedad de aplicaciones, desde aquellas que necesitan precisión del orden de las micras a aquellas que necesitan cientos de kilovatios de potencia. Entre sus usos se incluyen la robótica, maquinaria diversa, procesamiento de papel y madera, líneas de ensamblaje, procesamiento de alimentos, aplicaciones de impresión, embalaje, manipulación de materiales, soldadura automática, procesos de construcción en industria, corte por láser, etc.

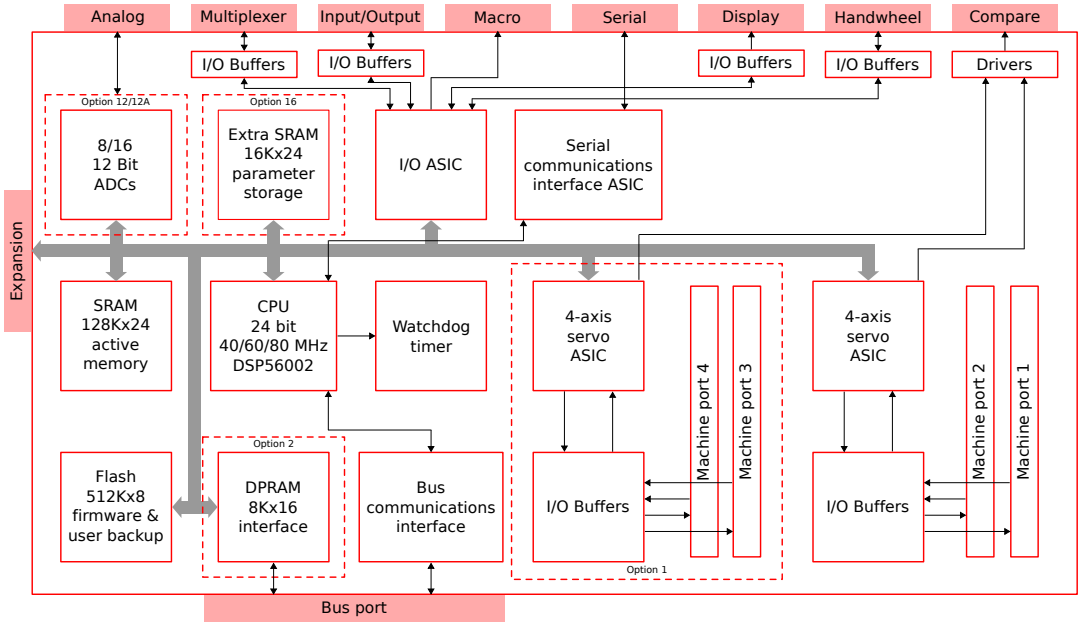

Figura 2.24: Diagrama de bloques de la tarjeta PMAC2-PCI.

Es importante decir que la tarjeta PMAC2-PCI es un computador completo, capaz de funcionar de manera autónoma con sus propios programas. Es más, esta tarjeta es un ordenador multitarea capaz de trabajar en tiempo real y priorizar actividades. Incluso cuando la PMAC2-PCI trabaja conjuntamente con un computador, las comunicaciones entre ambos elementos deben ser entendidas como entre dos computadores y no como entre un computador y un periférico. Un aspecto a destacar es la posibilidad de hacer trabajar hasta 16 PMAC2 de manera sincronizada para trabajar simultáneamente con 128 ejes.

La PMAC2-PCI dispone de múltiples modos de controlar los motores, sin embargo, no está diseñada para ser conectada directamente a los drivers que alimentan a los motores y a los encoders que se usan en los motores para averiguar en que posición se hayan estos. Dependiendo de

la forma en que se quiera comandar a los motores se deberá elegir un conjunto de tarjetas adicionales que hagan de interfaz entre la PMAC2-PCI y estos dispositivos. Estas tarjetas adicionales son ofrecidas también por Delta Tau Data Systems, Inc. Los modos de controlar los motores que nos ofrece la PMAC2-PCI son:

1. Control analógico mediante dos salidas analógicas de ± 10 V por motor, denominadas DAC, que permiten realizar los siguientes tipos de control (empleando los correspondientes drivers y accesorios):
   - Control analógico por velocidad.
   - Control analógico por par.
   - Control analógico mediante una onda senoidal.
2. Control digital mediante una señal digital modulada por ancho de pulso (PWM).
3. Control mediante una señal de pulso y dirección, para motores paso a paso.

Para capturar las señales necesarias para realizar el control de los motores, la tarjeta controladora dispone de:

- 3 entradas por motor para capturar las señales procedentes de los encoders.
- 4 entradas por motor para capturar diferentes señales que puedan ser de utilidad, como por ejemplo, señal de "home", señal de fallo del driver, etc.
- 5 entradas para señales de los sensores de efecto Hall.

Entre las herramientas programables que ofrece este modelo para realizar el control de los motores destacan:

1. Algoritmos de control PID que pueden modificarse manualmente o automáticamente mediante herramientas software.
2. Un lenguaje propio de programación que permite generar diversos tipos de programas (PLCs, programas de movimiento) que pueden ser descargados, guardados y ejecutados dentro de la memoria de la tarjeta controladora. Esta memoria es de tipo flash, lo que permite la conservación de los datos aún cuando la tarjeta controladora no es alimentada.

3. Diferentes tipos de interpolación para los movimientos: lineal, circular o mediante splines cúbicos.

4. La tarjeta controladora cuenta con 1025 variables para configurar su funcionamiento. Una vez que se ha configurado el comportamiento de la tarjeta dicha configuración se puede guardar en la memoria interna de la controladora.

La tarjeta PMAC2-PCI usada en Manfred necesita un interfaz adicional para comunicar con los drivers que alimentan los motores. Este interfaz adicional se llama "accesorio 8E". Se necesitan 4 de estas tarjetas para comandar los 8 motores de Manfred, ya que cada una de ellas tiene la lógica necesaria para interaccionar con dos de ellos.

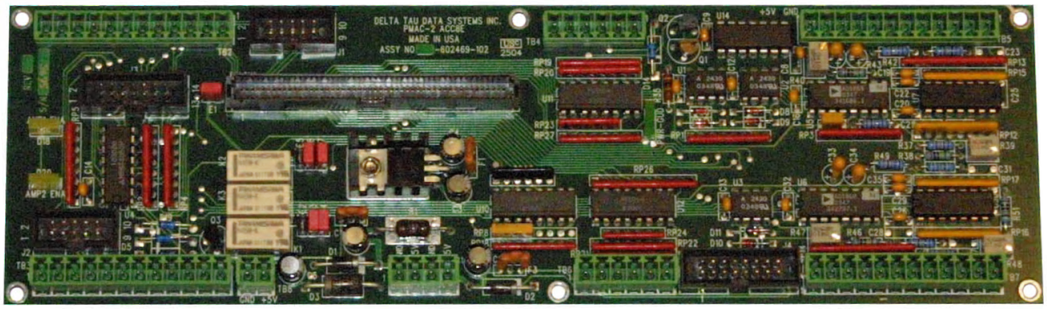

Figura 2.25: Vista general del accesorio 8E.

Cada accesorio 8E se conecta a la PMAC2-PCI a través de un bus plano de 100 pines denominado JMACH. Cada accesorio 8E dispone de cuatro conversores de 18-bit D/A para comandar dos drivers de entrada analógica. Cada accesorio 8E debe alimentarse con una tensión de ± 15 V. El accesorio 8E dispone también de entradas para la lectura de 2 encoder y 5 entradas por eje para poder capturar diferentes tipos de eventos como:

1. Señal de error.

2. Señal de home. En el robot Manfred esta señal es indispensable para los motores del brazo, ya que al usar encoder incrementales es necesario tener un dispositivo (un sensor inductivo en este caso) que avise cuando pasa el motor por un punto predeterminado. De esta forma, con esta señal se puede programar a la controladora para que lleve los 6 motores del brazo a una posición de inicio.

3. Dos señales para los limites del motor.

4. Una señal cualquiera a definir por el usuario. Esta señal permite capturar un evento externo en caso de que fuera necesario para alguna aplicación.

Los 8 drivers instalados en Manfred corresponden al modelo BE25A20 de la firma Advanced Motion Controls. Este driver es el encargado de conmutar las 3 fases del motor asociado para conseguir una velocidad de giro de su eje proporcional al nivel de tensión recibido del accesorio 8E. Para realizar adecuadamente la conmutación de las fases del motor se emplean sensores de efecto hall. El driver también es el encargado de aportar la alimentación necesaria al sensor inductivo y al encoder de cada motor. El esquema de conexión entre la PMAC, los accesorios 8E, los drivers, los motores y los encoders puede verse en la figura 2.27.

Figura 2.26: Vista general del driver BE25A20.

La configuración de la tarjeta PMAC2-PCI es una tarea muy laboriosa, y para ello se dispone de dos manuales, el software reference manual y el PMAC2 user manual, junto con un programa que proporciona el fabricante, de nombre PEWIN32 PRO, que se ejecuta bajo sistema operativo Windows (XP, Vista, 7).

El software PEWIN32 PRO ofrece un conjunto de herramientas que permiten modificar todos los parámetros de configuración de la PMAC2-PCI de una forma eficaz. Algunas de estas herramientas son:

- **El terminal:** Mediante esta herramienta se le pueden enviar comandos en codificación ASCII a la tarjeta controladora para que los ejecute.

- **Watch window:** Se trata de una ventana donde puede visualizar en tiempo real los valores de aquellas variables de la tarjeta controladora que se hayan elegido.
- **Tunning Pro:** Esta herramienta permite configurar parámetros de la PMAC2PCI tales como controladores PID, filtros, calibración de los DAC, etc.
- **Ventana de posición:** Esta ventana permite visualizar el valor en cuentas van tomando los motores, al igual que su velocidad, también en cuentas, y su error de seguimiento.

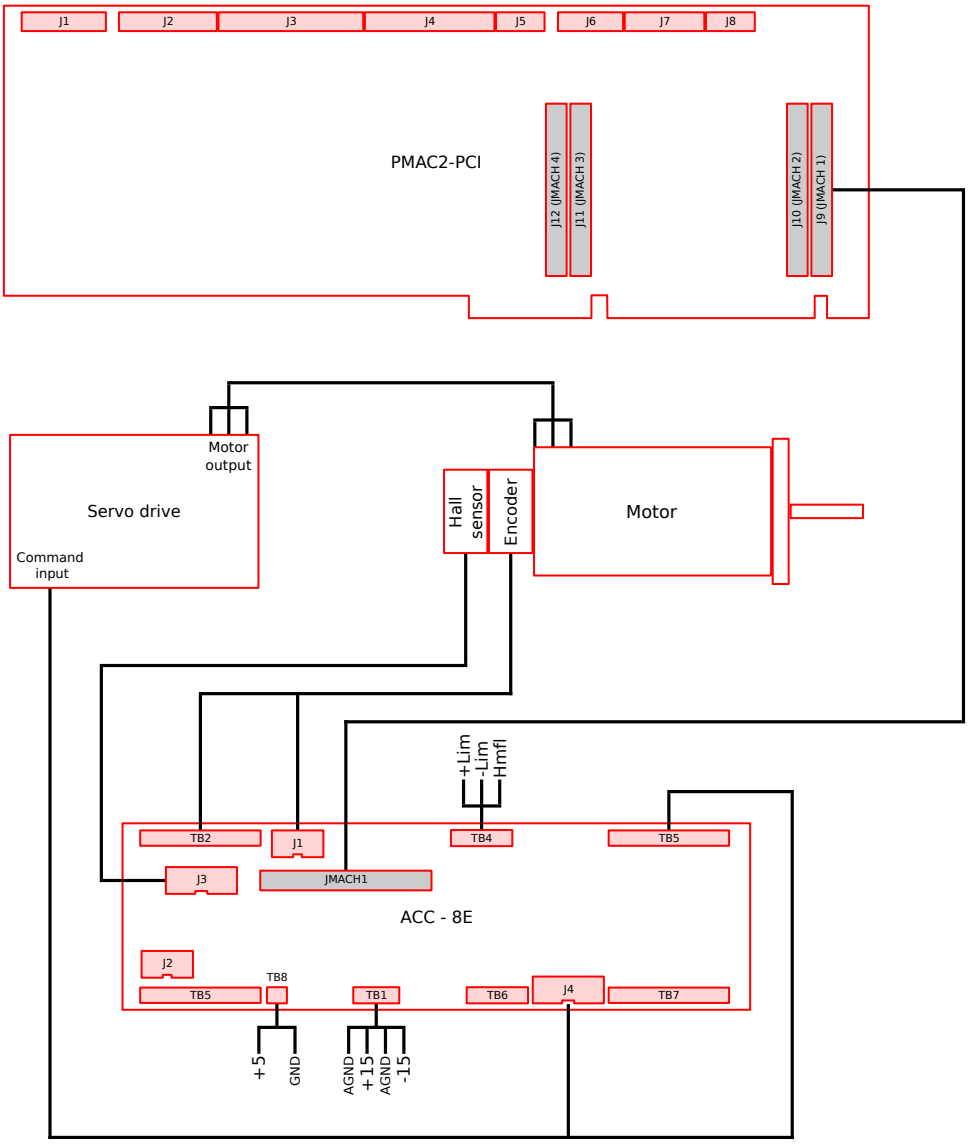

Figura 2.27: Esquema de conexionado entre la PMAC2-PCI, los accesorios 8E, los encoders y los motores del robot.

Las variables de configuración de la tarjeta controladora PMAC2-PCI se denominan I-variables. En total hay 1024 variables que configuran hasta el más mínimo detalle de la tarjeta controladora. El software PEWIN32 PRO es de gran ayuda a la hora de realizar los cambios pertinentes en la configuración de la tarjeta controladora gracias a sus intuitivas interfaces gráficas de usuario, aunque si se prefiere se puede realizar dicha configuración desde la aplicación términal, sin interfaces gráficas. La principal dificultad con la que se encuentra un usuario de esta tarjeta controladora es que muchas de las I-variables se encuentran relacionadas, de forma que la variación de una de ellas modifica el efecto otras.

La tarjeta controladora PMAC2-PCI puede ejecutar distintos tipos de movimiento, que pueden clasificarse en dos grandes grupos:

- **Movimientos de prueba (`jog`):** Un comando de tipo `jog` permite que un motor realice movimientos independientes de los movimientos que otros motores puedan estar realizando; es decir, son comandos que realizan movimientos no sincronizados de los motores. Por lo tanto el tiempo que emplea cada motor en completar su recorrido puede ser distinto, dependiendo de la velocidad a la que se mueva. Estos movimientos son recomendables cuando se realizan tareas de diagnóstico o depuración de código. También son los comandos adecuados cuando se quiere realizar control en velocidad de los motores ya que estos comandos permiten darle a los motores órdenes del estilo:
  Mover el motor 3 usando un perfil de velocidad cuadrático hasta que llegue a una velocidad final de $20^\circ/\mathrm{s}$ *y usando una aceleración lineal cuyo máximo, en valor absoluta, sea de* $5^\circ/\mathrm{s}^2$*. Tras alcanzar dicha velocidad final mantenerla hasta nueva orden.*

  Así pues, cuando se usan estos comandos es el desarrollador el que tiene que controlar que los movimientos de todos los motores concluyan a la vez, si este es el comportamiento deseado. Este tipo de comandos permite modificar en cualquier instante de tiempo la orden enviada a un motor. La nueva orden sobreescribe la anterior inmediatamente tras su envío al motor correspondiente.

- **Movimientos sincronizados (`pvt`, `linear`, `radius`, etc.):** Esta modalidad de trabajo se emplea cuando se quiere sincronizar el movimiento de varios de motores; es decir, el tiempo que emplean todos los motores en completar su recorrido es el mismo. Por lo tanto es tipo de comandos se usa cuando se quiere controlar un motor en posición. En este tipo de movimientos la secuencia de trabajo suele ser de la forma:

.....

Mueve el motor 3 y el motor 8 un ángulo de 45°*, usando un perfil de velocidad cuadrático cuya velocidad final sea de* 5°/s *y empleando un tiempo de* 1500 ms *en completar ambos movimientos.*

Espera 10 s.

Mueve el motor 3 y el motor 2 un ángulo de 15°*, usando un perfil de velocidad cuadrático cuya velocidad final sea de* 2°/s *y empleando un tiempo de* 500 ms *en completar ambos movimientos.*

.....

Los programas de movimiento son ejecutados línea a línea por la tarjeta controladora. Una vez finalizado el programa éste no volverá a ser ejecutado hasta que la tarjeta controladora reciba un comando que así lo indique. Desde un programa de movimiento sincronizado sólo es posible realizar llamadas a otros programas de movimiento sincronizado y realizar bucles de control de tipo `while`. En este tipo de programas no se puede enviar una nueva orden hasta que la anterior ha finalizado. Así la única manera de parar un movimiento antes de su terminación natural es abortarlo. La tarjeta controladora PMAC2-PCI tiene capacidad para almacenar y ejecutar hasta 256 programas de movimiento.

- **Programmable logic controller (PLC):** Los programas PLC nacen de la necesidad de tener programas que se ejecuten de forma continua, es decir, ejecutando continuamente sus tareas de forma asíncrona a los movimientos realizados por los motores. Estos programas se escriben de forma análoga a los programas de movimiento sincronizado, exceptuando que en la definición de su encabezamiento se definen como PLC. Una vez entran en ejecución, en cada ciclo de la tarjeta controladora, se ejecutará parte de dichos programas según la capacidad disponible por ésta.

2.4.1. DPRAM

La tarjeta PMAC2-PCI puede hacer uso de un accesorio optativo conocido como DPRAM (Dual-Ported RAM). Este dispositivo es básicamente una memoria RAM de 8 KPalabras con un tamaño de palabra de 16 bits, es decir, 16 KB.

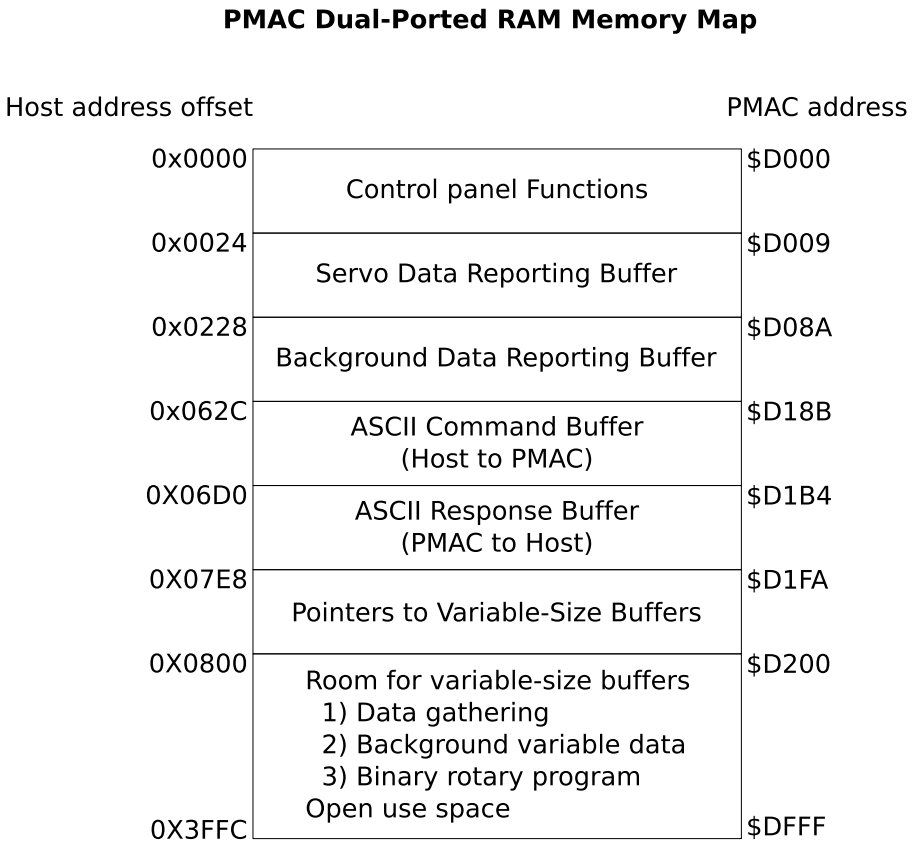

Figura 2.28: Disposición de los segmentos de memoria en el dispositivo DPRAM.

En la figura 2.28 puede observarse la tarea a la que se destina cada segmento de memoria que forma la DPRAM. La DPRAM es una tarjeta que tiene la mitad de tamaño que la PMAC2-PCI y que se instala justo encima de ella, quedando unidas ambas tarjetas con unos conectores metálicos y comunicadas por unos buses planos. Por medio de este dispositivo la tarjeta controladora PMAC2-PCI y el computador del robot intercambian información rápidamente. Una porción de la memoria RAM del ordenador de Manfred destinada a los buses PCI está mapeada a la DPRAM. Cualquier dato escrito en esa zona de la memoria RAM del PC de Manfred se transfiere automáticamente a la DPRAM y viceversa. El cometido principal de la DPRAM es el de agilizar la transferencia de datos entre la tarjeta y el ordenador.

La comunicación entre la tarjeta y el computador se realiza mediante comandos codificados en ASCII. La memoria comprendida entre las direcciones $D18B y $D1B3 se utiliza para realizar el intercambio de comandos ASCII entre el computador base y la tarjeta. Por otro lado la PMAC2-PCI deja el resultado de las operaciones solicitados por el computador base en el segmento de

memoria comprendido entre las direcciones $D1B4 y $D1F9. La memoria comprendida entre las direcciones $D200 y $DFFF permite una intercambio rápido bidireccional de variables.

Capítulo 3

Seguimiento de trayectorias empleando el algoritmo geométrico pure pursuit

Un algoritmo de seguimiento de trayectorias tiene como objetivo conseguir las leyes de control que permitan al robot seguir la ruta establecida de la forma más aproximada posible. Se pueden distinguir dos tipos de trayectorias. Aquellas que están constituidas por posturas de referencia, donde cada postura está formada por posición y orientación, y aquellas que están constituidas sólo por posiciones de referencia. En este proyecto sólo se han considerado estas últimas. En el problema del seguimiento se pretende que el error de orientación, θ_{err}, entre la orientación del robot, θ_r, y la orientación de la trayectoria, α, tienda a cero manteniendo acotadas las señales de control. Cabe destacar que el problema del seguimiento de una trayectoria involucra tanto el control de la dirección como el de la velocidad del vehículo autónomo.

Existen numerosos métodos de seguimiento, basados en teoría de control no lineal, control predictivo, linealización del modelo cinemático, métodos geométricos, etc. Dentro de los métodos geométricos, el más conocido y sencillo es el de persecución pura (pure pursuit en inglés). Este último método de control es el que ha usado para implementar un algoritmo de seguimiento de trayectorias en Manfred debido a su facilidad de implementación y a los buenos resultados que ofrece.

Otro aspecto a tener en cuenta es la representación de las trayectorias. Normalmente las trayectorias suelen presentarse como un conjunto de puntos de control por los que el robot debe pasar. Para ello si se dispone de una trayectoria continua en 2D ésta debe discretizarse, transformándose en una lista de puntos bidimensionales. Cada punto está posicionado en el

entorno usando coordenadas globales `x` e `y`. Para obtener una trayectoria discreta se toman muestras, preferiblemente equidistantes, de la trayectoria continua. Cuanto más próximas se encuentren las muestras más parecida es la trayectoria discreta a la trayectoria continua. Al igual que sucede con la trayectoria continua, la trayectoria discreta sólo puede ser recorrida en un sentido único, es decir, existe un orden estricto en el que recorrer los puntos de la trayectoria discreta. Así el punto i debe ser alcanzado por el robot antes que el punto $i + 1$. Se ha considerado que el segmento recto que une el punto i con el punto $i + 1$ sea referido como segmento i.

El algoritmo pure-pursuit tiene por objetivo hacer que el robot siga una trayectoria arbitraria asegurando un bajo error en posición y orientación. Para ello obliga al robot cada pocos segundos o milisengundos a calcular su ubicación y a calcular el radio de una circunferencia que éste debe seguir para corregir su error de orientación con respecto a la trayectoria de interés y conseguir así acercarse hacia un punto seleccionado de ésta que se encuentra a una cierta distancia por delante de él. A continuación se describen los pasos que ejecuta el algoritmo pure-pursuit:

- En primer lugar, el algoritmo debe conocer la ubicación absoluta del robot en el marco de referencia global. A partir de esta ubicación, y dado que también se conocen las coordenadas de los puntos de la trayectoria de interés en el marco de referencia global, el algoritmo calcula en que segmento de la trayectoria[1] se encuentra la proyección ortogonal de la posición del robot. Desde la posición proyectada en la trayectoria se busca, en el sentido de avance de ésta, el primero de sus puntos de control que se encuentre separado de la posición del robot una distancia L_{ah} en línea recta. A esta distancia se la ha designado con el símbolo L_{ah} debido a que en literatura anglosajona esta distancia es conocida con el nombre de *lookahead distance*. En el capítulo 6 se detalla el proceso de cálculo del valor de la distancia L_{ah}.

- A continuación el algoritmo pure-pursuit calcula el radio del arco que el robot debe seguir para ir desde su ubicación actual hasta el punto objetivo seleccionado en la trayectoria,

[1] Se entiende por segmento de la trayectoria aquel que une dos puntos de control consecutivos de ésta.

situado a una distancia L_{ah} por delante suya. Ver figura 3.1.

$$\Delta x = x_{ah} - x_r \tag{3.1}$$

$$\Delta y = y_{ah} - y_r \tag{3.2}$$

$$L_{ah} = \sqrt{\Delta x^2 + \Delta y^2} \tag{3.3}$$

$$= \sqrt{x_{ah}^{'\,2} + y_{ah}^{'\,2}} \tag{3.4}$$

El error de orientación se puede calcular fácilmente observando el dibujo:

$$\tan\alpha = \frac{\Delta y}{\Delta x} \tag{3.5}$$

$$\alpha = \theta_r + \theta_{err} \tag{3.6}$$

$$\theta_{err} = \alpha - \theta_r = \arctan\left(\frac{\Delta y}{\Delta y}\right) - \theta_r \tag{3.7}$$

A continuación se calcula el radio del arco que debe trazar el robot:

$$R_s = d + y_{ah}^{'} \tag{3.8}$$

$$R_s^2 = d^2 + x_{ah}^{'\,2} \tag{3.9}$$

$$= \left(R_s - y_{ah}^{'}\right)^2 + x_{ah}^{'\,2} \tag{3.10}$$

Desarrollando la expresión 3.10 se obtiene:

$$2 \cdot R_s \cdot y_{ah}^{'} = L_{ah}^2 \tag{3.11}$$

Teniendo en cuenta que $y_{ah}^{'} = L_{ah} \cdot \sin(\theta_{err})$ la expresión 3.11 se puede simplificar, quedando:

$$R_s = \frac{L_{ah}}{2 \cdot \sin(\theta_{err})} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{2 \cdot \sin\left(\arctan\left(\frac{\Delta y}{\Delta x}\right) - \theta_r\right)} \tag{3.12}$$

Como se puede observar en la expresión 3.12 el algoritmo pure-pursuit sólo cuenta con un sólo parámetro de control, la distancia L_{ah}. Por este motivo el algoritmo es fácilmente implementable y configurable. El comportamiento del robot depende del valor escogido para el parámetro L_{ah}. Así, cuanto menor es el valor otorgado a dicha variable más preciso es el seguimiento que el robot realiza de la trayectoria, menor es el radio del arco que describe el robot y más *agresivo* es el retorno del robot a la trayectoria cuando se aparta de ella (presencia de oscilaciones repetidas). A pesar de estas características existen buenas razones

para aumentar el valor de este parámetro. Cuando el valor del parámetro L_{ah} aumenta el número de oscilaciones que sufre el robot al seguir la trayectoria disminuye y se suavizan, no son tan abruptas como en el caso anterior. Este comportamiento hace que el robot comience a girar antes en aquellos lugares donde la trayectoria tiene cambios muy pronunciados (picos angulosos), dando lugar a caminos más suaves. Este comportamiento es especialmente importante en vehículos que operan a una alta velocidad ya que las trayectorias suaves aumentan la estabilidad, y por ende la seguridad, con la que el robot se desplaza. Además, cuanto es mayor es el valor de la distancia L_{ah} más suave es la aproximación del robot a la trayectoria.

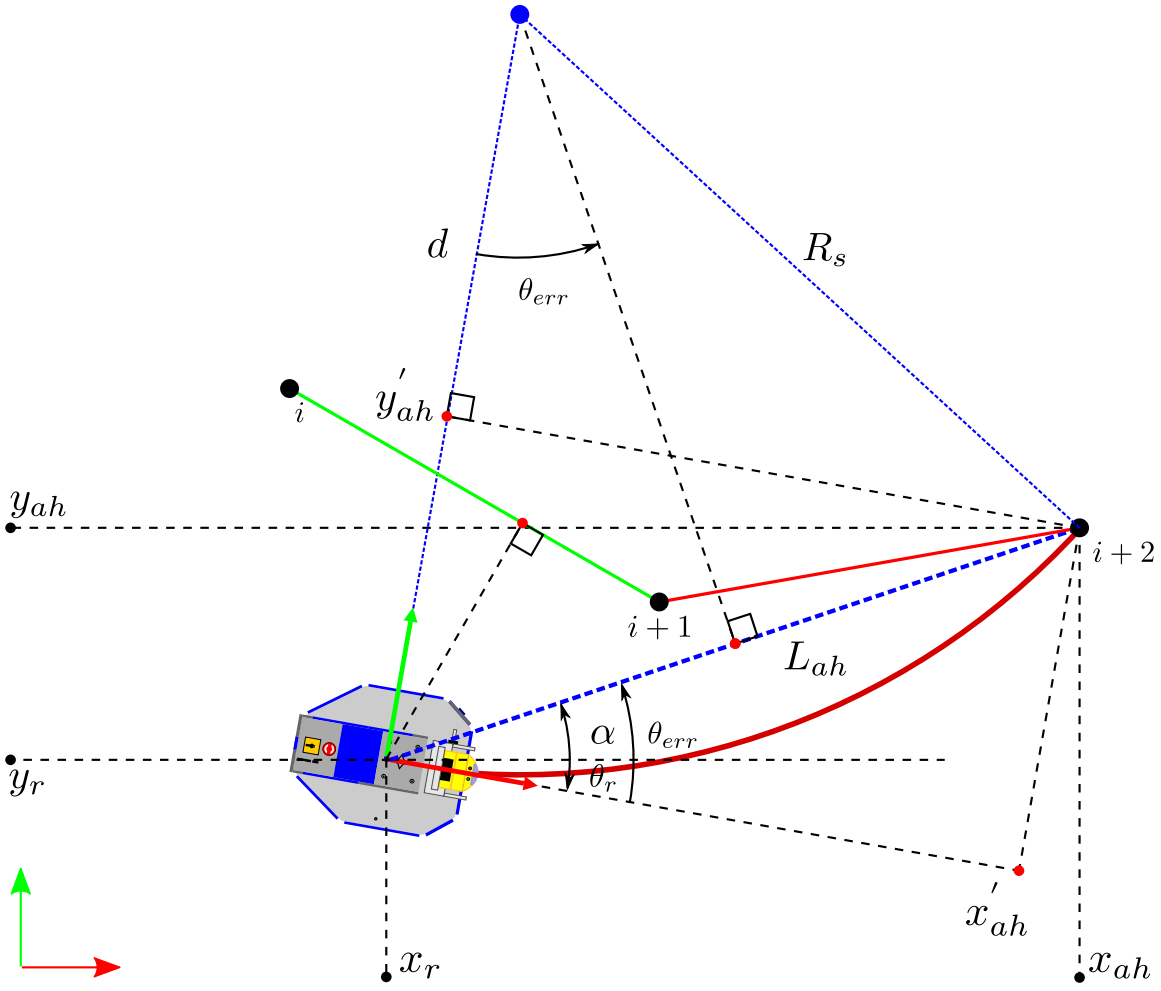

Figura 3.1: Análisis geométrico del algoritmo pure-pursuit.

- Durante un breve intervalo de tiempo el robot se desplaza trazando el arco cuyo radio se

calculó en el punto anterior. Pasado el período de tiempo mencionado el algoritmo pure-pursuit debe disponer de la ubicación absoluta del robot en el marco de referencia del mundo. Por tanto, en el robot se debe ejecutar un algoritmo de localización absoluta que comunique al algoritmo pure-pursuit la ubicación del robot en el marco de referencia global. Ver trabajo [1].

- A continuación empieza un nuevo ciclo de trabajo. El algoritmo pure-pursuit a partir de la posición absoluta del robot calcula su proyección ortogonal en la trayectoria. Seguidamente se obtiene un nuevo punto objetivo en la trayectoria situado a una distancia L_{ah} por delante del robot. Se calcula el nuevo radio de giro que debe usar el robot para trazar un arco que corrija su orientación y que lo encamine hacia el nuevo punto objetivo, y así sucesivamente hasta que se alcance el punto final de la trayectoria, momento en el que el robot finaliza su desplazamiento y queda a la espera de más órdenes.

3.1. Búsqueda de la proyección ortogonal de la posición del robot en la trayectoria

Según se desplaza el robot, siguiendo la trayectoria de interés discreta con la mayor exactitud posible, se determina el primer segmento de ésta por el que aún no haya pasado el robot en el que intersecte de manera perpendicular el segmento que pasa por el punto medio de la línea imaginaria que une los centros de ambas ruedas. En el punto medio de la línea imaginaria que une los centros de ambas ruedas se ha definido un marco de referencia denominado `link_base`. El punto de intersección de ambos segmentos define la proyección ortogonal de la posición del robot en la trayectoria de interés. De ahora en adelante, el segmento de la trayectoria discreta donde se encuentra la proyección ortogonal de la posición del robot se conocerá como segmento relevante. El segmento relevante es el punto de partida para encontrar el siguiente punto objetivo del algoritmo pure pursuit. En el resto de la sección se describirá el procedimiento geométrico de búsqueda usado para determinar si un segmento de la trayectoria es relevante o no. El procedimiento de búsqueda comprueba si el segmento seleccionado es relevante y en caso de que no lo sea se comprueba el segmento contiguo, y si tampoco es relevante, se continua la búsqueda con el siguiente segmento y así sucesivamente se van comprobando uno a uno los segmentos de la trayectoria hasta encontrar el primero que sea relevante. Una vez que se ha encontrado el primer

segmento relevante comienza la búsqueda del próximo punto objetivo del algoritmo pure pursuit. El proceso de búsqueda del siguiente segmento relevante comenzará en el último segmento relevante encontrado, es decir, hay solapamiento inicial del segmento de búsqueda.

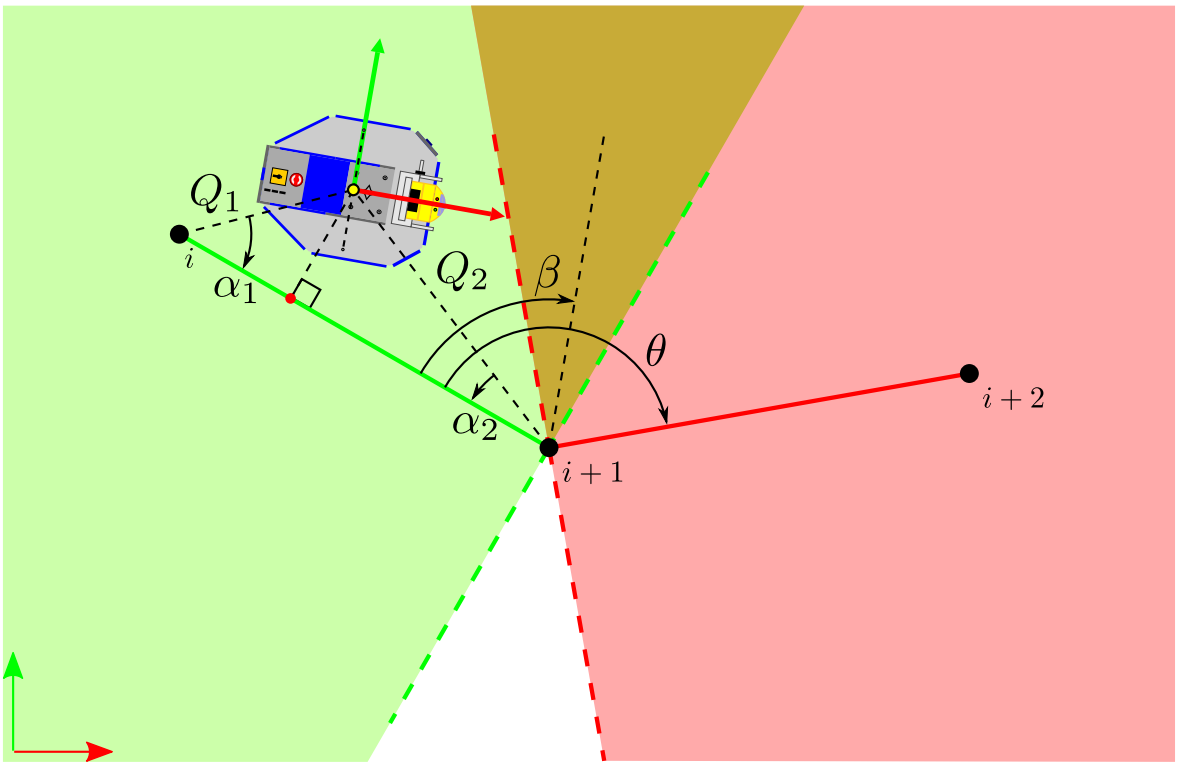

Figura 3.2: Parámetros geométricos usados para analizar la trayectoria. En marrón claro aparece la zona de solapamiento. En blanco aparece la zona muerta o zona de no solapamiento.

La determinación de si un segmento de la trayectoria es relevante requiere de cálculo de varios parámetros. Observe la figura 3.2, el segmento Q_1 une el punto i con el marco de referencia `link_base`. El segmento Q_2 une el punto $i+1$ con el marco de referencia `link_base`. Los ángulos que se forman desde Q_1 y Q_2 hasta el segmento i se denominan α_1 y α_2 respectivamente. Por último el ángulo que se forma desde el segmento i hasta el segmento contiguo $i+1$ se denomina θ. Los ángulos recorridos en sentido horario se consideran negativos, mientras que aquellos que son recorridos en sentido antihorario son considerados positivos. El ángulo θ se encuentra en el intervalo $[-\pi, \pi]$. En cambio, los ángulos α_1 y α_2 se encuentran en el intervalo $[-2\pi, 2\pi]$.

Para hallar el valor de los ángulos α_1 y α_2 es necesario calcular en antes el valor de los ángulos

γ y λ. Ver figura 3.3.

$$\gamma = \arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right) \tag{3.13}$$

$$\lambda = \arctan\left(\frac{y_i - y_{i+1}}{x_i - x_{i+1}}\right) = \gamma - sign\,(\gamma) \cdot \pi \tag{3.14}$$

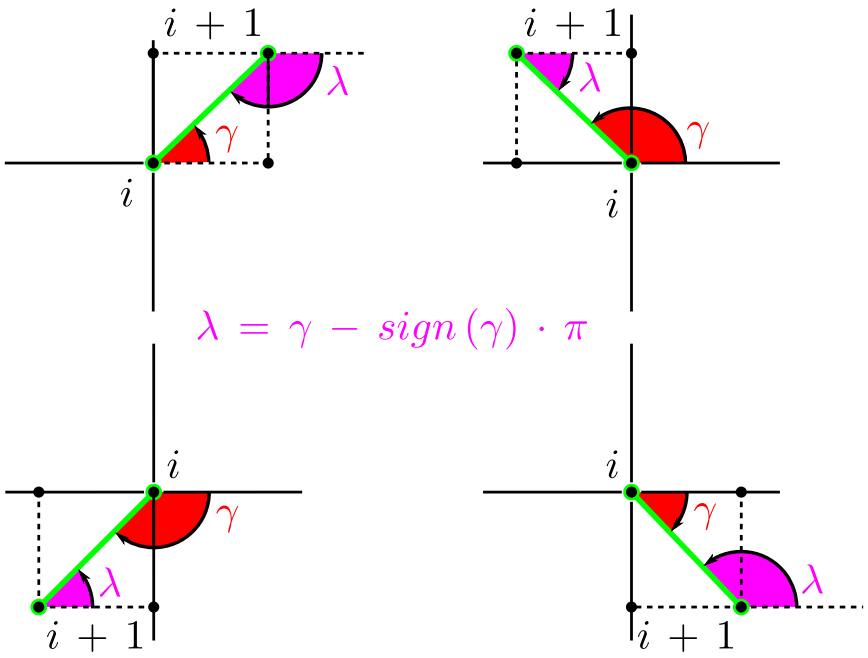

Figura 3.3: Ángulos formados por el segmento i e $i + 1$con la horizontal.

$$\alpha_1 = \gamma - \arctan\left(\frac{y_r - y_i}{x_r - x_i}\right) \tag{3.15}$$

$$\alpha_2 = \lambda - \arctan\left(\frac{y_r - y_i}{x_r - x_i}\right) \tag{3.16}$$

$$\theta = \arctan\left(\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}}\right) - \lambda \tag{3.17}$$

$$\text{Si } |\theta| > \pi \longrightarrow \theta = \theta - sign\,(\theta) \cdot \pi$$

$$\beta = \frac{\theta}{2} \tag{3.18}$$

El caso más sencillo de determinación de un segmento relevante de trayectoria se ilustra en la figura 3.4.

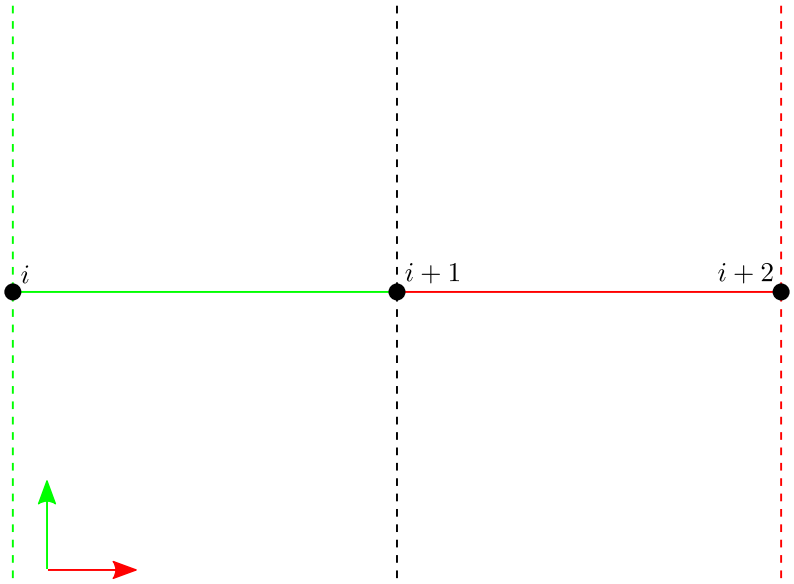

Figura 3.4: Regiones de relevancia para una trayectoria en linea recta.

En este caso los segmentos de la trayectoria están dispuestos uno tras otro en línea recta y las líneas perpendiculares discontinuas marcan la región del espacio en la que cada segmento es relevante. Si el robot se encuentra entre las líneas fronterizas discontinuas de los nodos i e $i+1$, entonces el segmento i es el segmento relevante y por tanto la posición del robot se proyecta perpendicularmente sobre este segmento.

Analizando la figura 3.4 se llega a la conclusión de que los términos $|\alpha_1|$ y $|\alpha_2|$ son los que determinan si un segmento es relevante o no. Si la magnitud de ambos ángulos es menor o igual que $\frac{\pi}{2}$ entonces el segmento analizado es relevante y por tanto contiene la proyección perpendicular del origen del marco de referencia `link_base`. Si la magnitud de alguno de esos ángulos es superior a $\frac{\pi}{2}$ entonces el segmento analizado no es relevante.

Sin embargo, la situación descrita en la figura 3.4 es poco habitual, y en trayectorias cuyos segmentos no son colineales determinar si un segmento es relevante es más complejo.

En la figura 3.2 se puede observar como las lineas fronterizas discontinuas perpendiculares a los segmentos de la trayectoria generan una zona de solapamiento en el interior del trazado y una zona no-solapamiento o zona muerta en el exterior del trazado.

Cuando el robot se encuentra en el interior de una curva, como en la figura 3.2, se considera que el segmento relevante es el i si el robot se haya antes de la línea bisectriz ($\beta = \frac{\theta}{2}$) que separa ambos segmentos. En cambio el segmento relevante es el $i+1$ si el robot ha cruzado la línea

bisectriz. De este modo se elimina la ambigüedad que pudiera existir en la zona de solapamiento. Analizando la figura 3.2 se obtiene que el robot debe cumplir una de las dos condiciones siguientes para hayarse en el interior de una curva:

$$\alpha_2 \leq 0 \text{ y } \beta > 0 \tag{3.19}$$

$$\alpha_2 \geq 0 \text{ y } \beta < 0 \tag{3.20}$$

A partir de las condiciones anteriores se pueden obtener cuatros posibles casos de estudio:

Caso 1: El robot se encuentra en el interior de una curva.

En primer lugar se cumple que $|\alpha_1| \leq \frac{\pi}{2}$. También se satisface una de las siguientes condiciones, o la 3.19 o la 3.20. Si además el segmento i es relevante se verifica que:

$$|\alpha_2| \leq |\beta| \tag{3.21}$$

Si la condición 3.21 no se cumple, entonces el robot se encuentra en el interior de una curva, ha cruzado la bisectriz que delimita la región de influencia del segmento i y se ha adentrado en la región de influencia del segmento $i+1$. Por tanto el segmento relevante es el $i+1$. En la figura 3.2 puede observar un ejemplo en el que las condiciones 3.20 y 3.21 se cumplen.

Caso 2: El robot se encuentra en el exterior de una curva.

De nuevo, en esta situación se verifica que $|\alpha_1| \leq \frac{\pi}{2}$, pero fallan las condiciones 3.19 y 3.20. Si se cumple que $|\alpha_2| \leq \frac{\pi}{2}$ entonces el segmento i es relevante. En cambio, si $|\alpha_2| > \frac{\pi}{2}$ entonces habrá que comprobar la magnitud del ángulo α_1 del segmento $i+1$ para determinar si el robot se encuentra en la zona muerta o se elige como segmento relevante el segmento $i+1$.

Caso 3: El robot se encuentra en la zona muerta.

En la figura 3.5 se puede observar al robot situado en la zona muerta entre los segmentos i y $i+1$.

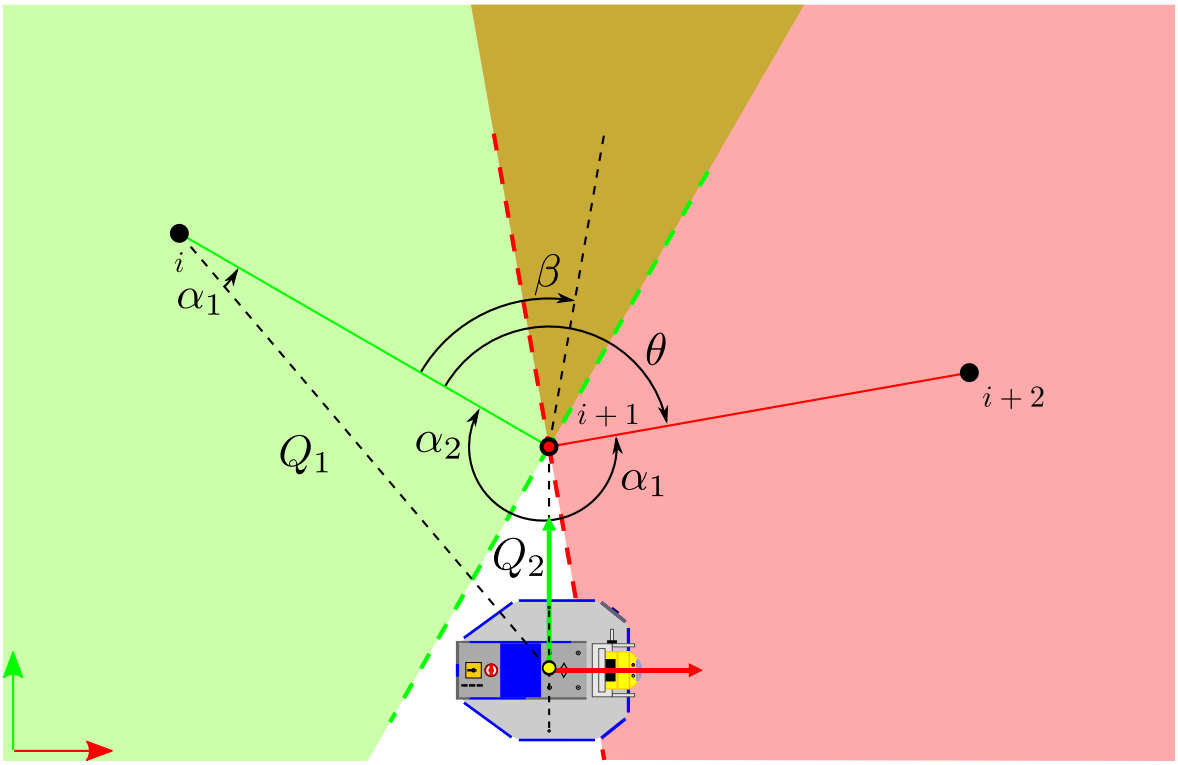

Figura 3.5: Robot situado en la zona muerta existente entre dos segmentos consecutivos de la trayectoria.

De nuevo se cumple la condición $|\alpha_1| \leq \frac{\pi}{2}$ y no cumple la condición 3.19 y tampoco la condición 3.20. El segmento i no es relevante ya que $|\alpha_2| > \frac{\pi}{2}$. Para que el robot se encuentre en la zona muerta, una vez que se han cumplido las condiciones anteriores, el ángulo α_1 del segmento $i+1$ debe cumplir que $|\alpha_1| > \frac{\pi}{2}$. Cuando el robot se haya en zona muerta su posición no se proyecta sobre un segmento sino que se proyecta sobre un punto, el punto $i+1$. Por otro lado, si el ángulo α_1 del segmento $i+1$ cumple que $|\alpha_1| \leq \frac{\pi}{2}$, entonces el robot ha abandonado la zona muerta y se ha adentrado en la zona de influencia del segmento $i+1$, por tanto el segmento relevante es el $i+1$.

Caso 4: El robot se encuentre por detrás del primer nodo de la trayectoria.

Finalmente, si el ángulo α_1 que se consigue con el primer segmento de la trayectoria cumple que $|\alpha_1| > \frac{\pi}{2}$, entonces el primer segmento de la trayectoria se haya delante del robot. En estas circunstancias y dependiendo de las necesidades de la aplicación el robot puede actuar de dos maneras diferentes. Bien rotar sobre sí mismo el ángulo necesario para alinearse con

el primer punto de la trayectoria y a continuación ejecutar el algoritmo pure-pursuit o bien obviar la rotación inicial y ejecutar directamente el algoritmo pure-pursuit para ir aproximándose a la trayectoria.

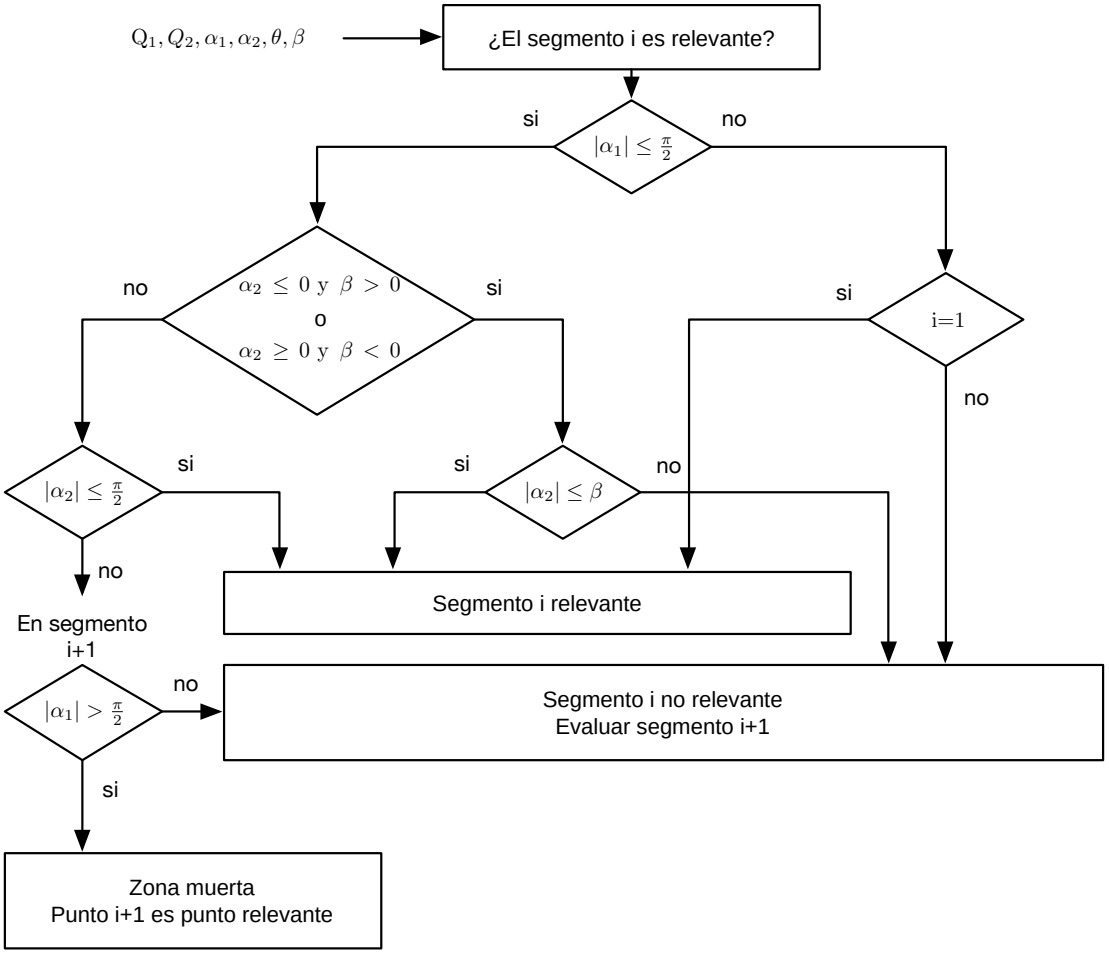

Figura 3.6: Diagrama de flujo del algoritmo pure-pursuit.

Capítulo 4

Odometría en el robot Manfred

4.1. ¿Qué es la odometría?

La palabra odometría proviene de las raíces griegas *odos* y *metron* que significan *camino* y *medición*, respectivamente. Se atribuye a Herón de Alejandría la creación del primer aparato para determinar la distancia viajada por los carruajes durante un trayecto a partir de la cuenta de las revoluciones de sus ruedas. Este artefacto fue llamado *odómetro* y constituye el predecesor de los actuales velocímetros analógicos de los automóviles.

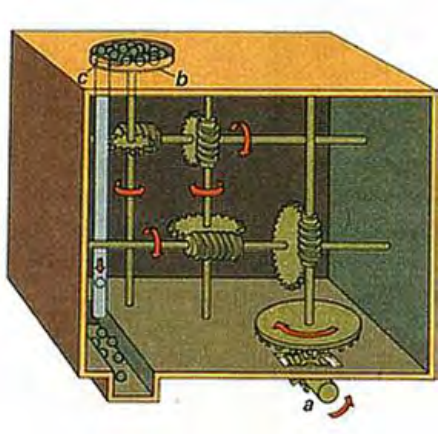

Figura 4.1: Odómetro de Herón de Alejandría.

Su principio de funcionamiento era sencillo, varios engranajes se movían, impulsados por la fuerza de la rueda principal del vehículo o del carro. Esto hacía rodar una plataforma circular que contenía un pequeño agujero y en la que se acumulaban numerosas bolas. Cuando esta plataforma

circular giraba las bolitas caían por un tubo hasta una caja en la que volvían a amontonarse. Para medir los kilómetros recorridos sólo había que contar las bolitas que habían caído y hacer los cálculos precisos. Se ha demostrado que era un sistema bastante exacto. Por ejemplo, la medición con odómetro entre las ciudades de Hecatompylos y Alejandría fue de 850 km, cuando en realidad es de 852 km.

Actualmente bajo el nombre de odometría se engloba a un conjunto diverso de técnicas que se utilizan para estimar la ubicación de un objeto móvil con respecto a un sistema de referencia. Se habla de estimación de la ubicación ya que saber con total exactitud la ubicación de un robot móvil es imposible. ¿Por qué? La única forma que posee un robot móvil de obtener información de su entorno es a través de sensores que proporcionan medidas que no están exentas de errores (los errores de los sensores son inversamente proporcionales al precio de los mismos). A partir de las medidas proporcionada por esos sensores y usando modelos matemáticos, que en ocasiones deben ser ligeramente simplificados, se obtiene la ubicación deseada. Tanto los errores de adquisición en las medidas como las simplificaciones realizadas de los modelos matemáticos conllevan una reducción en la precisión del resultado, por lo cual se considera que este valor no es más que una estimación de la ubicación verdadera del robot. Normalmente la discrepancia entre la estimación y la ubicación verdadera está acotada por debajo de un valor que se considera máximo admisible en función de las necesidades de la aplicación desarrollada.

En el caso de un robot móvil que se desplaza en un espacio bidimensional la odometría debe estimar la posición y orientación del robot; es decir, el vector $(x\,(t_k)\,,\, y\,(t_k)\,,\, \theta\,(t_k))^T$. El término k simplemente es un número entero positivo que representa un índice de adquisición o muestreo. Por tanto t_k representa el instante de tiempo en el que se adquiere la muestra k de la ubicación del robot.

En robótica las ubicaciones del robot móvil se expresan en el sistema de referencia del mundo (punto característico del entorno de trabajo); recibiendo a veces el nombre de ubicaciones absolutas. En cambio, las ubicaciones estimadas por los cálculos odométricos no están expresadas con respecto al sistema de referencia del mundo, sino con respecto al sistema de referencia que posee el robot en el momento de iniciar dichos cálculos, esto es, el sistema de referencia `link_base`. Es importante enfatizar que cuando el robot se pone en marcha por primera vez la ubicación del sistema de referencia `link_base` en el marco de referencia global no es conocida. Y por este motivo al mismo tiempo que se inicia la odometría del robot se inicia un algoritmo de localización absoluta que estimará las coordenadas del marco de referencia `link_base` en el marco global.

A partir de estas coordenadas y de las coordenadas proporcionadas por la odometría se puede hacer una predicción de la ubicación del robot.

4.2. Dispositivos sensibles al movimiento: encoders

Los encoders son dispositivos electromecánicos que generan señales digitales en respuesta al movimiento. Están disponibles en dos tipos, uno que responde a la rotación, y el otro al movimiento lineal. A su vez cada uno de los tipos anteriores están disponibles en dos categorías. Una es el encoder incremental, que genera pulsos mientras se mueve el elemento mecánico al que está asociado. Se utilizan para medir directamente velocidad y aceleración, aunque también permiten obtener posición siempre que se conozca el posición de partida del elemento mecánico asociado. El otro tipo es el encoder absoluto, que genera un código binario que indica directamente la posición actual del elemento mecánico asociado. Los encoders pueden utilizar tanto tecnología óptica como magnética. El sensor óptico provee altas resoluciones, velocidades de operaciones muy altas, y ofrece una larga vida en la mayoría de los ambientes industriales. Los sensores magnéticos se utilizan frecuentemente en aplicaciones de trabajo pesado, donde son frecuentes el polvo, la humedad, los golpes y las vibraciones. Este tipo de sensores proveen buena resolución, altas velocidades de operación y son muy resistentes.

En las ruedas motrices de la base del robot Manfred se usan encoders ópticos rotatorios incrementales. Estos encoders se hayan acoplados al eje del motor de las ruedas y permiten obtener la posición, velocidad y aceleración del motor.

En el contexto de la robótica este sensor se clasifica dentro de la categoría de los sensores propioceptivos, que son aquellos que proporcionan información sobre el estado interno del robot, en este caso el estado en uno de sus actuadores.

Un encoder óptico rotativo incremental utiliza un disco de vidrio o plástico opaco con un patrón de ranuras depositadas en su perímetro exterior equitativamente espaciadas Este disco se haya iluminado por un led infrarrojo. Al otro lado del disco existe un fotodetector. El receptor tiene la tarea de detectar el paso de la luz infrarroja a través de los espacios presentes en el disco. Cada vez que una ranura se alinea con el led emisor se detecta luz en el fotoreceptor y se obtiene un pulso eléctrico digital a la salida del encoder.

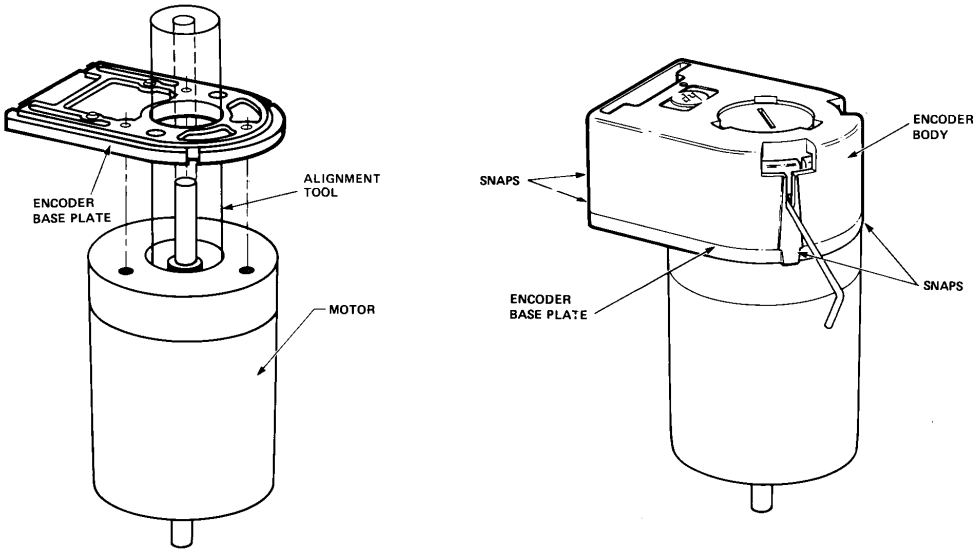

Figura 4.2: Colocación de un encoder óptico rotatorio incremental en un motor.

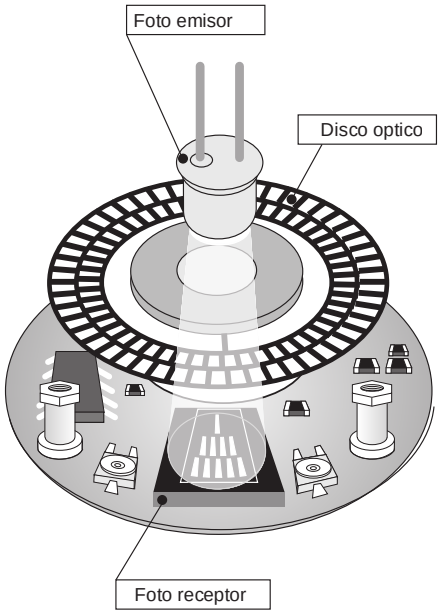

Figura 4.3: Interior de un encoder óptico rotativo incremental.

Por ejemplo los encoders ópticos rotativos incrementales usados en las ruedas de Manfred poseen un disco con 500 ranuras, es decir, se detectan 500 pulsos por revolución (ppr) de cada

rueda. La resolución del encoder[1] es:

$$\Delta\phi = \frac{360^\circ}{500} = 0'720^\circ = 0'013\,\text{rad} \tag{4.1}$$

Cuando se utiliza un fotoreceptor se puede conocer el desplazamiento de la rueda pero no se puede conocer el sentido de su giro; es decir, el encoder es unidireccional. Si se quiere conocer también el sentido de su giro se debe usar un encoder óptico rotativo incremental que posea dos conjuntos de ranuras, desplazadas entre sí la distancia equivalente a media ranura y dos fotoreceptores. Ver disco perforado en la figura 4.3. Con esta disposición de los elementos se generan dos señales de pulsos digitales desfasadas 90° ($\pi/2$ rad), de ahí que este encoder suela recibir el nombre de encoder en cuadratura. A estas señales de salida se les llama comúnmente A y B. Mediante ellas es posible suministrar información de posición angular, velocidad angular, aceleración angular y dirección de rotación del eje. Es muy habitual que se incluya una tercera señal electrónica digital de salida, denominada índice (I), que genera un pulso por cada revolución completa del disco.

Supongamos un encoder con 500 ppr. Las señales digitales A y B que se obtendrán con el transcurso del tiempo son:

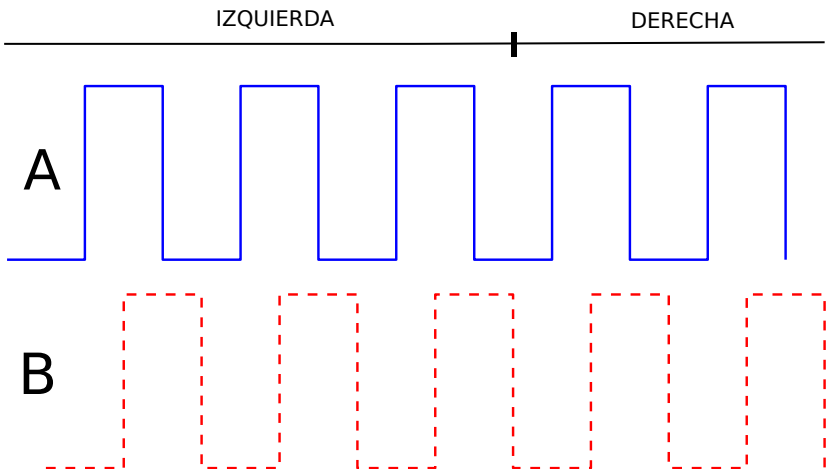

Figura 4.4: Canales A y B de un encoder óptico rotativo incremental.

Con cada revolución completa del encoder se generan 500 pulsos digitales en la señal A y en la señal B. Obviamente cada pulso digital cuenta con un flanco de subida (transición de nivel bajo a nivel alto) y un flanco de bajada (transición de nivel alto a nivel bajo). Para determinar

[1] Ángulo mínimo rotado por la rueda que se puede detectar.

el ángulo girado por el encoder se cuenta el número de flancos de subida (o de bajada) generados en una de las señales, bien en A o en B. La dirección vendrá determinada, en cada pulso, por el nivel de la otra señal, de tal forma que si éste es un nivel bajo se determina que el sentido de giro es uno concreto, en cambio si el nivel es alto el sentido de giro es el opuesto al anterior. Ejemplos:

- Si se monitoriza la señal digital A y se obtiene un flanco de subida.
  - Si en B se obtiene un nivel bajo: Desplazamiento a la derecha.
  - Si en B se obtiene un nivel alto: Desplazamiento a la izquierda.
- Si se monitoriza la señal digital A y se obtiene un flanco de bajada.
  - Si en B se obtiene un nivel bajo: Desplazamiento a la izquierda.
  - Si en B se obtiene un nivel alto: Desplazamiento a la derecha.
- Si se monitoriza la señal digital B y se obtiene un flanco de subida.
  - Si en A se obtiene un nivel bajo: Desplazamiento a la izquierda.
  - Si en A se obtiene un nivel alto: Desplazamiento a la derecha.
- Si se monitoriza la señal digital B y se obtiene un flanco de bajada.
  - Si en A se obtiene un nivel bajo: Desplazamiento a la derecha.
  - Si en A se obtiene un nivel alto: Desplazamiento a la izquierda.

Se puede mejorar la resolución del encoder en un factor 2 o 4 contando los dos flancos de subida (o de bajada) de las señales A y B o contando los cuatro flancos de la señal A y B (dos de bajada y dos de subida) respectivamente. Siguiendo con el ejemplo, un encoder en cuadratura de 500 ranuras es equivalente a un encoder en cuadratura con un disco de 2000 ranuras en el que sólo se tiene en cuenta, para medir el desplazamiento, el flanco de subida o el de bajada de una de las señales. Ejemplo: Se registran los flancos de subida la señal A. Por cada flanco de subida registrado la rueda ha rotado $0'72°$. Se hace la suposición de que el desplazamiento es hacia la derecha, por tanto el nivel de la señal B es bajo. Si además se registra el flanco de subida de la señal B entonces el ángulo rotado por la rueda es $0'72° + (90°/360°) \cdot 0'72° = 0'90°$ Si la rueda sigue girando y se registra el flanco de bajada de la señal A entonces el ángulo rotado por la

rueda es: $0'72° + (180°/360°) \cdot 0'72° = 1'08°$. Si la rueda sigue girando y se registra el flanco de bajada de la señal B entonces el ángulo rotado por la rueda es: $0'72° + (270°/360°) \cdot 0'72° = 1'26°$. Finalmente, si la rueda sigue girando y se registra otro flanco de subida de la señal A, entonces se ha completado un período de la señal A y la rueda ha rotado un ángulo de $1'44°$.

4.3. Odometría de la base diferencial del robot Manfred

Dependiendo del número de ruedas motrices de un robot móvil y de cómo se dispongan éstas en su base se obtiene un conjunto diferente de expresiones para estimar la ubicación del robot. Existen numerosas formas de disponer las ruedas motrices en una base móvil. Una de las más habituales es la *base diferencial*. En esta disposición el robot cuenta con dos ruedas motrices, paralelas entre sí, controladas individualmente. En esta disposición cada rueda puede moverse únicamente en dos sentidos, hacia delante o hacia atrás. Esta disposición es fácilmente implementable y sencilla de analizar. Por estos motivos la base diferencial es muy adecuada para la navegación en entornos típicamente humanos, por ejemplo oficinas, laboratorios, etc. Junto a estas ruedas suelen disponerse otras, pasivas, denominadas ruedas locas cuyo objetivo es dotar de estabilidad a la base del robot. La base diferencial es con diferencia la disposición que proporciona unos cálculos odométricos más sencillos. La traslación y rotación de las plataformas diferenciales se determinan por la relación del movimiento independiente de cada una de sus ruedas motrices. Por ejemplo si en la base diferencial ambas ruedas giran hacia delante con la misma velocidad el robot se desplaza en línea recta de frente. En cambio, si ambas ruedas giran hacia atrás a la misma velocidad el robot se desplaza en línea recta hacia atrás. Si las ruedas giran en sentidos opuestos pero con la misma velocidad el robot rota alrededor del eje perpendicular a la línea imaginaria que une el centro de ambas ruedas y que pasa por el punto medio de dicha línea. La rotación es horaria si la rueda derecha gira hacia atrás y la rueda izquierda gira hacia delante. Por otro lado, la rotación es antihoraria si la rueda derecha gira hacia delante y a la rueda izquierda gira hacia atrás. Para cualquier otra combinación de giros de las ruedas el movimiento de traslación resultante del robot no es tan sencillo de visualizar y se puede entender a partir del estudio de las expresiones odométricas que se detallan más adelante en esta sección. La velocidad

de desplazamiento lineal de la rueda derecha e izquierda se obtienen de la forma:

$$v_d\left(t_k\right) = \omega_d\left(t_k\right) \cdot r_r \tag{4.2}$$

$$v_i\left(t_k\right) = \omega_i\left(t_k\right) \cdot r_r \tag{4.3}$$

donde los términos $\omega_d\left(t_k\right)$ y $\omega_i\left(t_k\right)$ representan las velocidades angulares de las ruedas derecha e izquierda respectivamente (expresadas en rad/s normalmente) y el término r_r representa el radio de las ruedas.

Por otro lado, de manera alternativa, la velocidad de desplazamiento lineal de la rueda derecha, izquierda y del punto medio de la línea imaginaria que une el centro de ambas ruedas (velocidad de desplazamiento lineal del robot) se obtienen de la forma:

$$v_d\left(t_k\right) = \left(R_s\left(t_k\right) + r_b\right) \cdot \omega_b\left(t_k\right) \tag{4.4}$$

$$v_i\left(t_k\right) = \left(R_s\left(t_k\right) - r_b\right) \cdot \omega_b\left(t_k\right) \tag{4.5}$$

$$v_b\left(t_k\right) = R_s\left(t_k\right) \cdot \omega_b\left(t_k\right) \tag{4.6}$$

donde $\omega_b\left(t_k\right)$ representa la velocidad angular instantánea con la que el robot gira alrededor un punto, denominado centro de rotación instantáneo (CRI). El término $R_s\left(t_k\right)$ es el radio instantáneo de traslación circular, alrededor del CRI, del punto medio de la línea imaginaria que une el centro geométrico de ambas ruedas. Este término será positivo si el robot gira en sentido antihorario alrededor del CRI y negativo si gira en sentido horario. El término r_b es la mitad de la distancia que separa el centro geométrico de las ruedas de la base móvil, es decir, el radio de la base móvil.

Las ecuaciones cinemáticas del punto medio de la línea imaginaria que une el centro geométrico de ambas ruedas son:

$$\dot{x}\left(t_k\right) = v_b\left(t_k\right) \cdot \cos\left(\theta\left(t_k\right)\right) \tag{4.7}$$

$$\dot{y}\left(t_k\right) = v_b\left(t_k\right) \cdot \sin\left(\theta\left(t_k\right)\right) \tag{4.8}$$

$$\dot{\theta}\left(t_k\right) = \omega_b\left(t_k\right) \tag{4.9}$$

La posición y orientación del robot móvil se obtiene integrando las velocidades de éste en un

periodo de tiempo de duración indefinida.

$$x\left(t_k\right) \;=\; x\left(t_{k-1}\right) \;+\; \Delta x_k \;=\; x\left(t_{k-1}\right) \;+\; \int_{t_{k-1}}^{t_k} v_b\left(\tau\right) \cdot \cos\left(\theta\left(\tau\right)\right) d\tau \tag{4.10}$$

$$y\left(t_k\right) \;=\; y\left(t_{k-1}\right) \;+\; \Delta y_k \;=\; y\left(t_{k-1}\right) \;+\; \int_{t_{k-1}}^{t_k} v_b\left(\tau\right) \cdot \sin\left(\theta\left(\tau\right)\right)\, d\tau \tag{4.11}$$

$$\theta\left(t_k\right) \;=\; \theta\left(t_{k-1}\right) \;+\; \Delta\theta_k \;=\; \theta\left(t_{k-1}\right) \;+\; \int_{t_{k-1}}^{t_k} \omega_b\left(\tau\right)\, d\tau \tag{4.12}$$

La integral de la expresión 4.12 suele tener solución analítica, sin embargo, las integrales que aparecen en la expresiones 4.10 y 4.11 suelen carecer de ella, por tanto se hace obligatorio encontrar otro método que permita obtener de manera aproximada sus resultados. Entre estos métodos se encuentran las integraciones numéricas, de diferente tipo: integración numérica por el método del trapecio, integración numérica de Simpson 1/3, integración numérica de Simpson 3/8, etc. A pesar de que en el párrafo anterior se ha indicado que la integral de la expresión 4.12 suele tener solución analítica, la expresión matemática que describe el término $w_b\left(t_k\right)$ no siempre es conocida a priori, y por tanto hay que ir muestreando dicha función a través de las medidas que realizan los encoders. Si el periodo de observación, $\Delta t_k \;=\; t_k \;-\; t_{k-1}$, tiende a 0, entonces la integral de la expresión 4.12 puede ser aproximada con una precisión adecuada usando la regla de la integración por trapecio:

$$\Delta\theta_k \;=\; \int_{t_{k-1}}^{t_k} \omega_b\left(\tau\right)\, d\tau \;\approx\; \omega_b\left(t_k\right) \;\cdot \Delta t_k \tag{4.13}$$

Para mantener el período de observación a un valor próximo a 0, la frecuencia de adquisición de la información de los encoders de las ruedas deber ser muy elevada, típicamente cada pocos ms. El objetivo de los siguientes desarrollos son relacionar el cambio de orientación del robot, $\Delta\theta_k$, en primer lugar con las velocidades angulares de las ruedas del mismo, $\omega_d\left(t_k\right)$ y $\omega_i\left(t_k\right)$, y en última instancia con las cuentas que registran los encoders de ambas ruedas, ΔC_{dk} y ΔC_{ik}, durante el periodo de muestreo Δt_k.

Para relacionar el término $\Delta\theta_k$ con las velocidades angulares de las ruedas del robot en primer lugar se restan las expresiones 4.4 y 4.5:

$$\omega_b\left(t_k\right) \;=\; \frac{v_d\left(t_k\right) \;-\; v_i\left(t_k\right)}{2 \cdot r_b} \tag{4.14}$$

En segundo lugar en el resultado anterior se sustituyen las expresiones 4.2 y 4.3:

$$\omega_b\left(t_k\right) \;=\; \frac{1}{2} \cdot \frac{r_r}{r_b} \cdot \left(\omega_d\left(t_k\right) \;-\; \omega_i\left(t_k\right)\right) \tag{4.15}$$

Sustituyendo la expresión 4.15 en la expresión 4.13 se puede expresar el término $\Delta\theta_k$ en función de las velocidades angulares de las ruedas:

$$\Delta\theta_k \approx \omega_b(t_k) \cdot \Delta t_k \tag{4.16}$$

$$\approx \frac{1}{2} \cdot \frac{r_r}{r_b} \cdot (\omega_d(t_k) - \omega_i(t_k)) \cdot \Delta t_k \tag{4.17}$$

Las cuentas que registran los encoders de ambas ruedas durante el periodo de muestreo Δt_k, considerando que en este intervalo de tiempo las velocidades de las ruedas se han mantenido constantes con valores $\omega_d(t_k)$ y $\omega_i(t_k)$, se obtienen con las expresiones:

$$\Delta C_{dk} = Q \cdot \omega_d(t_k) \cdot \Delta t_k \tag{4.18}$$

$$\Delta C_{ik} = Q \cdot \omega_i(t_k) \cdot \Delta t_k \tag{4.19}$$

donde el término Q es un factor de conversión de radianes a cuentas. Este término puede calcularse fácilmente anotando primero el número de cuentas registradas por uno de los encoders cuando la rueda a la que está vinculado completa una vuelta y en segundo lugar realizando el cociente entre este número y $2 \cdot \pi$ (número de radianes que constituyen una vuelta de la rueda). Por lo tanto la expresión 4.17 pueden ser expresada en función de estos términos como:

$$\Delta\theta_k \approx \frac{1}{2 \cdot Q} \cdot \frac{r_r}{r_b} \cdot (\Delta C_{dk} - \Delta C_{ik}) \tag{4.20}$$

Otro término de especial interés, cuya expresión se quiere conocer, es $R_s(t_k)$. Despejando dicho término de la expresión 4.6 se obtiene:

$$R_s(t_k) = \frac{v_b(t_k)}{\omega_b(t_k)} \tag{4.21}$$

Para expresar el término $v_b(t_k)$ en función de las velocidades angulares de las ruedas se debe, en primer lugar, sumar las expresiones 4.4 y 4.5 y tener en cuenta en dicha suma la expresión 4.6:

$$2 \cdot v_b(t_k) = v_d(t_k) + v_i(t_k) \tag{4.22}$$

En segundo lugar se deben sustituir en la expresión anterior las expresiones 4.2 y 4.3 y despejar el término $v_b(t_k)$:

$$v_b(t_k) = \frac{r_r}{2} \cdot (\omega_d(t_k) + \omega_i(t_k)) \tag{4.23}$$

Sustituyendo las expresiones 4.23 y 4.15 en la expresión 4.21 se puede despejar el término $R_s\left(t_k\right)$ en función de las velocidades angulares de las ruedas:

$$R_s\left(t_k\right) = \frac{v_b\left(t_k\right)}{\omega_b\left(t_k\right)} = r_b \cdot \left(\frac{\omega_d\left(t_k\right) + \omega_i\left(t_k\right)}{\omega_d\left(t_k\right) - \omega_i\left(t_k\right)}\right) \tag{4.24}$$

Si se multiplica el numerador y el denominador de la expresión 4.25 por el término $Q \cdot \Delta t_k$ se consigue expresar el término $R_s\left(t_k\right)$ en función de las cuentas registradas por los encoders durante el intervalo de tiempo Δt_k:

$$R_s\left(t_k\right) = r_b \cdot \left(\frac{\Delta C_{dk} + \Delta C_{ik}}{\Delta C_{dk} - \Delta C_{ik}}\right) \tag{4.25}$$

Si durante el período de tiempo Δt_k la velocidad de ambas ruedas es igual, tanto en magnitud como en sentido de giro, entonces el robot describe una trayectoria rectilínea. Por tanto no es necesario resolver integrales para conocer la ubicación del robot en el instante t_k. Dependiendo del sentido de giro de las ruedas, la trayectoria rectilínea es de frente o marcha atrás. En estas circunstancias se cumple que:

$$\omega_d\left(t_k\right) = \omega_i\left(t_k\right) \tag{4.26}$$

$$\omega_b\left(t_k\right) = 0 \text{ rad/s} \tag{4.27}$$

$$\Delta\theta_k = 0 \text{ rad} \tag{4.28}$$

$$\theta\left(t_k\right) = \theta\left(t_{k-1}\right) \tag{4.29}$$

$$R_s\left(t_k\right) = \infty \text{ m} \tag{4.30}$$

Nótese que el valor del radio de giro durante el intervalo de tiempo Δt_k es infinito; una recta se puede interpretar como una circunferencia de radio infinito. La distancia recorrida en línea recta por cada rueda es:

$$d = r_r \cdot \omega_d\left(t_k\right) \cdot \Delta t_k = r_r \cdot \frac{\Delta C_{dk}}{Q} \tag{4.31}$$

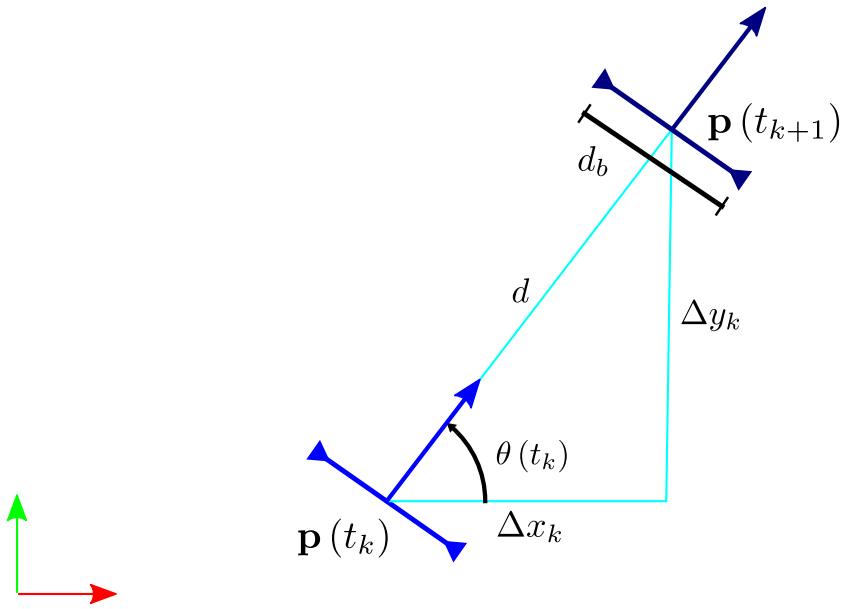

Figura 4.5: Desplazamiento del robot en línea recta.

Por tanto la variación de posición del robot en esta situación se puede expresar con las ecuaciones:

$$\Delta x_k = d \cdot \cos\left(\theta\left(t_k\right)\right) = r_r \cdot \frac{\Delta C_{dk}}{Q} \cdot \cos\left(\theta\left(t_k\right)\right) \tag{4.32}$$

$$\Delta y_k = d \cdot \sin\left(\theta\left(t_k\right)\right) = r_r \cdot \frac{\Delta C_{dk}}{Q} \cdot \sin\left(\theta\left(t_k\right)\right) \tag{4.33}$$

A continuación se estudiará el caso en el que durante el periodo Δt_k las magnitudes de las velocidades angulares de las ruedas motrices son distintas pero poseen el mismo sentido de giro. En estas circunstancias el robot sigue una trayectoria cualquiera, bien hacia delante, o bien marcha atrás. Para estimar la ubicación del robot en estas circunstancias se hace la consideración matemática de que la velocidad angular en cada rueda motriz se mantiene constante durante el período de muestreo Δt_k. Para que esta consideración sea aplicable éste intervalo de tiempo debe poseer un valor muy próximo a 0. Con esta consideración la trayectoria real del robot en el pequeño intervalo de tiempo Δt_k se puede aproximar por una trayectoria circular alrededor del CRI, con radio de giro $R_s\left(t_k\right)$. Ver figura 4.6. (Obviamente se hace la consideración de que la función radio de giro es constante durante el intervalo Δt_k ya que este último término se ha considerado que tiene un valor muy próximo a 0). Las diferentes trayectorias de aproximación que se pueden obtener con este criterio durante el intervalo de tiempo Δt_k son:

1. $\omega_d(t_k) > 0, \omega_i(t_k) > 0, \omega_d(t_k) > \omega_i(t_k)$: Traslación circular antihoraria de frente.

2. $\omega_d(t_k) > 0, \omega_i(t_k) > 0, \omega_d(t_k) < \omega_i(t_k)$: Traslación circular horaria de frente.

3. $\omega_d(t_k) < 0, \omega_i(t_k) < 0, |\omega_d(t_k)| > |\omega_i(t_k)|$: Traslación circular horaria marcha atrás.

4. $\omega_d(t_k) < 0, \omega_i(t_k) < 0, |\omega_d(t_k)| < |\omega_i(t_k)|$: Traslación circular antihoraria marcha atrás.

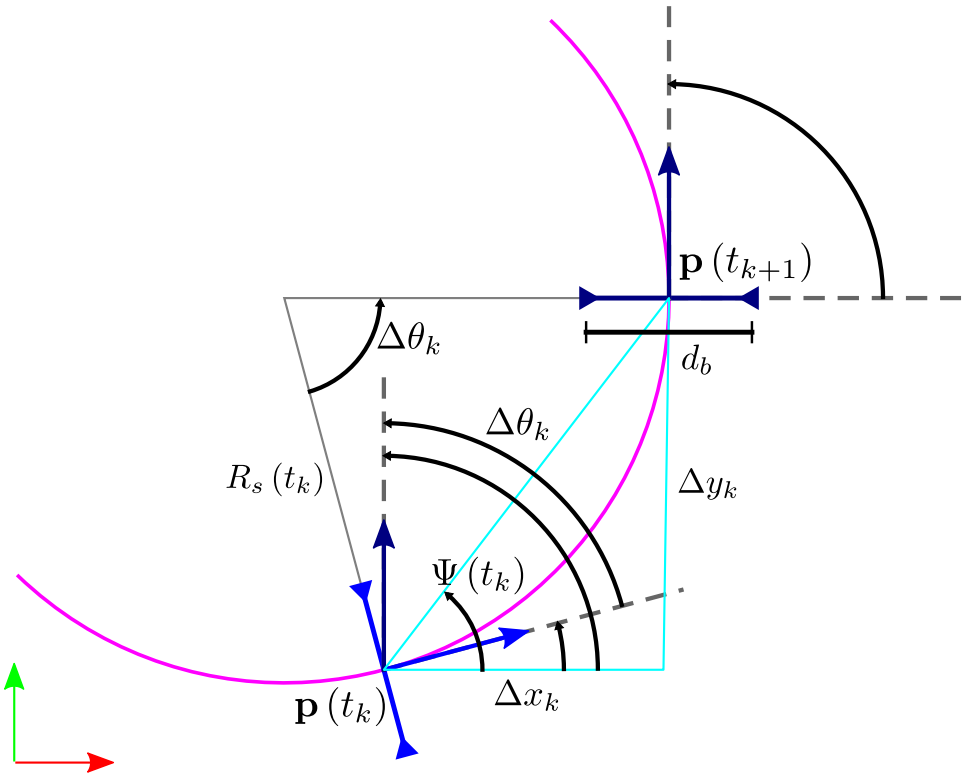

Figura 4.6: Desplazamiento del robot con una trayectoria circular.

Observado la figura 4.6 se pueden obtener las siguientes expresiones, que son válidas en todos los casos listados anteriormente:

$$\Psi(t_k) = \theta(t_{k-1}) + \frac{\Delta\theta_k}{2} \tag{4.34}$$

$$Cuerda = 2 \cdot R_s(t_k) \cdot \sin\left(\frac{\Delta\theta_k}{2}\right) \tag{4.35}$$

$$\Delta x_k \approx Cuerda \cdot \cos(\Psi(t_k)) \tag{4.36}$$

$$\approx 2 \cdot R_s(t_k) \cdot \sin\left(\frac{\Delta\theta_k}{2}\right) \cdot \cos\left(\theta(t_{k-1}) + \frac{\Delta\theta_k}{2}\right) \tag{4.37}$$

$$\Delta y_k \approx Cuerda \cdot \sin\left(\Psi\left(t_k\right)\right) \tag{4.38}$$

$$\approx 2 \cdot R_s\left(t_k\right) \cdot \sin\left(\frac{\Delta\theta_k}{2}\right) \cdot \sin\left(\theta\left(t_{k-1}\right) + \frac{\Delta\theta_k}{2}\right) \tag{4.39}$$

Si las magnitudes de las velocidades angulares de las ruedas motrices permanecen constantes durante el intervalo Δt_k entonces la situación real coincide con la consideración matemática y por tanto las aproximaciones ($\approx$) de las expresiones anteriores se transforman en igualdades ($=$), ya que el robot, en efecto, se traslada siguiendo un movimiento circular alrededor del CRI. El último caso de estudio, un caso particular del anterior, es aquel en el que el robot rota sobre sí mismo (alrededor del eje vertical perpendicular a la base del robot y que pasa por el punto medio de la línea imaginaria que une el centro de ambas ruedas). Para que el robot rote sobre sí mismo las magnitudes de las velocidades angulares de la ruedas motrices deben ser iguales, pero con sentidos de giro opuestos. Dependiendo de la relación de estos sentidos, la rotación es horaria o antihoraria, como se explicó anteriormente. En esta situación existirá variación en la orientación del robot, $\Delta\theta_k \neq 0$, mientras que su variación en posición será nula, $(\Delta x_k, \Delta y_k) = (0, 0)$, puesto que no se traslada del lugar que ocupa. Estos resultados también se pueden calcular usando las expresiones 4.20, 4.37 y 4.39. De este modo cuando el robot rota sobre sí mismo se obtiene:

$$\Delta C_{dk} = -\Delta C_{ik} \tag{4.40}$$

$$\Delta\theta_k \approx \frac{1}{Q} \cdot \frac{r_r}{r_b} \cdot \Delta C_{dk} \tag{4.41}$$

$$R_s\left(t_k\right) = 0\,\mathrm{m} \tag{4.42}$$

$$\Delta x_k \approx 0\,\mathrm{m} \tag{4.43}$$

$$\Delta y_k \approx 0\,\mathrm{m} \tag{4.44}$$

Por último hay que mencionar que si las magnitudes de las velocidades angulares de la ruedas motrices se mantienen constantes durante el intervalo de tiempo Δt_k, al igual que en el caso anterior, las aproximaciones se convierten en igualdades.

4.4. Limitaciones de la odometría

La odometría proporciona una estimación muy próxima a la ubicación real del robot a cortas distancias, es barato y permite una frecuencia de muestreo muy alta. Sin embargo, la idea

fundamental de la odometría es la integración de información incremental del movimiento de las ruedas a lo largo del tiempo, lo que conduce inevitablemente a la acumulación de errores. En particular, la acumulación de errores de orientación causa grandes errores de posición que aumentan con la distancia recorrida por el robot. A pesar de estas limitaciones, la mayoría de los investigadores coinciden en que la odometría es una parte importante del sistema de navegación del robot. Es utilizado en casi todos los robots móviles por diversas razones:

- En algunas ocasiones, la odometría es la única información disponible para navegar, por ejemplo, cuando no hay referencia externas o balizas que puedan ser detectadas por el robot o cuando otros sensores no puedan proporcionar datos utilizables.

- La información puede combinarse con la estimación de la ubicación absoluta del robot para obtener una predicción de la siguiente ubicación global que ocupará éste. Si el algoritmo de localización global deja de localizar temporalmente al robot, por ejemplo al no lograr su convergencia, se puede usar la odometría como mecanismo de localización absoluta. Obviamente este método introduce errores en la ubicación del robot con el paso del tiempo. Estos errores serán subsanados en cuanto el algoritmo de localización global recupere la convergencia.

La odometría se basa en sensores de bajo coste y ecuaciones simples que son fáciles de implementar en software. Sin embargo, también se basa en el supuesto de que las ruedas rotan, lo que se transforma en una traslación del robot. Esta suposición tiene una validez limitada. Un ejemplo es el deslizamiento de las ruedas sobre algunas superficies que ofrecen una baja adherencia de éstas. El resultado de este fenómeno es una estimación de la ubicación del robot errónea tras realizar los cálculos vistos anteriormente.

Los errores que afectan a la odometría suelen ser de muy diversa naturaleza y se clasifican en dos categorías, los errores sistemáticos y los errores no sistemáticos. El error sistemático, como su nombre indica, es aquel que se repite constantemente, es propio del sistema y se produce de igual modo en todas las medidas que se realizan de una magnitud. Puede estar originado por un defecto en cualquiera de los componentes que constituyen el robot. Entre los errores sistemáticos más habituales encontramos:

- Utilizar ruedas con diámetro diferente.

- Mala alineación de las ruedas.

- Incertidumbre del punto de contacto efectivo de las ruedas con el suelo (diámetro de la base, d_b).

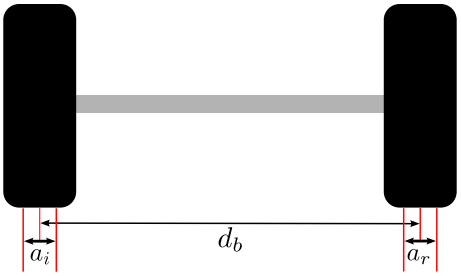

Figura 4.7: Los términos a_i y a_r representan la incertidumbre en el punto de contacto de las ruedas con el piso. Diferentes valores de a_i y a_r conducen a diferentes valores del diámetro de la base, d_b.

- Resolución finita del encoder. Cuanto mayor sea la resolución menor es el error introducido por este fenómeno.

Los errores sistemáticos son graves porque se acumulan constantemente. Los errores no sistemáticos son aquellos que tienen una naturaleza aleatoria, impredecible y por tanto pueden afectar en cualquier lugar y momento. En muchas ocasiones estos errores están provocados por el deslizamiento de las ruedas debido a:

- Suelos resbaladizos.
- Sobre-aceleración.
- Giros rápidos.
- Mal contacto de las ruedas en el suelo.
- Interacción con objetos externos.

En la mayoría de las superficies lisas los errores sistemáticos son mas peligros que los no sistemáticos, por contra, en las superficies rugosas, los errores sistemáticos son escasos y predominan los errores no sistemáticos. Debido a estos errores a medida que el robot se desplaza aumenta la región de incertidumbre que rodea a las sucesivas estimaciones de su ubicación que se calculan haciendo uso de la odometría. Ver figura 4.8. Esta región crece hasta que se obtiene la ubicación absoluta del robot, haciendo uso de algún algoritmo de localización global.

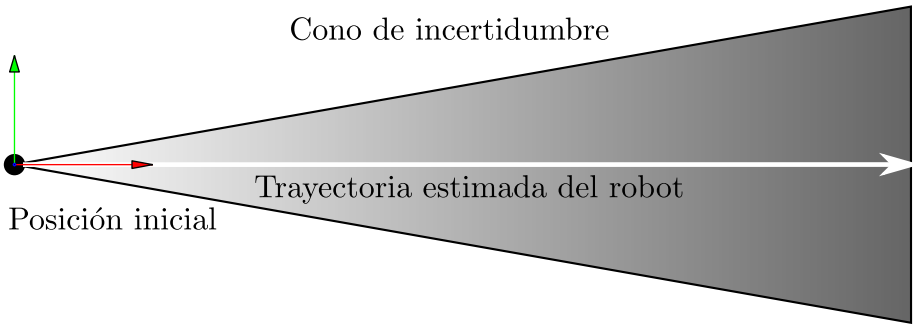

Figura 4.8: El cono de incertidumbre crece (y se oscurece) indicando que en la odometría se van acumulando errores que impiden conocer con total exactitud cual es la ubicación real del robot.

En resumen, la odometría no es una forma fiable de localizar a un robot móvil. Proporciona una forma sencilla de estimar su ubicación, pero esa estimación va acumulando errores, tanto sistemáticos como no sistemáticos. Eso quiere decir que en trayectorias largas (entendiendo como tal aquellas que superan los 15 o 20 m) la estimación de la ubicación por odometría deja de ser fiable. Como consecuencia, se debe corregir la estimación odométrica periódicamente con algún algoritmo de localización global para realizar un seguimiento adecuado del robot en su desplazamiento por el entorno.

Capítulo 5

El framework Robot Operating System

En la primera sección de este capítulo se presenta una visión general del framework robótico ROS en la que se describen los conceptos principales que lo constituyen sin entrar en detalle sobre su programación. También se indican cuales son las ventajas de usar este framework robótico. En la segunda sección se describe en detalle los paquetes más importantes que se han usado para desarrollar este proyecto.

ROS es un meta-sistema operativo[1] de código libre que posee numerosas características, de entre las cuales cabe citar:

1. Abstracción de hardware.
2. Control de dispositivos a bajo nivel.
3. Paso de mensajes entre procesos.
4. Gestión de paquetes
5. Posee herramientas y librerías para ejecutar procesos distribuidos en múltiples máquinas. Los procesos ejecutados en ROS se hayan débilmente acoplados gracias a los mecanismos de comunicación que implementa dicho framework. Estos mecanismos se basan en la comunicación síncrona al estilo RPC, de hecho es XML-RPC, comunicaciones asíncronas y almacenamiento compartido de datos en un servidor de parámetros al que tienen acceso todos los nodos de un sistemas escrito con ROS.

[1] Se comporta como un sistema operativo y se ejecuta sobre un sistema operativo real, por ejemplo Ubuntu.

Cabe destacar que ROS no es un framework de tiempo real aunque sí es posible integrar algunas partes de ROS con código que se deba ejecutar en tiempo real.

El objetivo principal de ROS es proporcionar un framework de programación de código abierto y unificado destinado al control de robots en multitud de entornos, ya sean virtuales o reales. Este software robótico nació en Willow Garage [2], una empresa californiana ubicada en Menlo Park y actualmente es mantenido por la Open Source Robotics Foundation (OSRF)

Willow Garage además desarrolla el robot PR2, figura 5.1, que incorpora el software de control ROS, con el propósito de fomentar el uso de dicho software. Algunas unidades del robot PR2 han sido prestadas a instituciones científicas y Universidades de todo el mundo, en virtud de acuerdos de colaboración mutua. El resto de unidades se comercializan a un precio que ronda los $400,000. El robot PR2 incorpora, entre otras cosas, dos cámaras stereo, un par de scanners láser, dos brazos con 7 grados de libertad, un sistema de locomoción omnidireccional, etc.

Figura 5.1: Robot PR2 desarrollado por Willow Garage

Actualmente ROS incluye software para multitud de tareas, entre las cuales cabe citar navegación, localización (SLAM), reconocimiento de objetos 3D, planificación de acciones, control de movimiento para brazos multi-articulados, aprendizaje máquina, etc.

[2]Willow Garage es un laboratorio de investigación robótica fundado en 2006 por Scott Hassan, reputado ex-empleado de Google y Standford.

Otra de la máximas del framework ROS es *no reinventes la rueda, no dupliques el trabajo que hizo otro*. Muchos científicos e ingenieros han estado escribiendo software para plataformas robóticas durante los últimos 50 años y ROS se ha propuesto aunar todo ese esfuerzo en un sólo lugar. Afortunadamente la Web es el medio perfecto para compartir código.

ROS cuenta con una enorme comunidad de desarrollo, debido a su enfoque didáctico y abierto, que se compone de investigadores, aficionados, fabricantes de hardware y empresas que le dan soporte como Willow Garage, la Open Source Robotics Foundation (OSRF) o incluso Google.

Todos estos actores pueden compartir sus repositorios de código de una manera sencilla y a coste cero situándolos en servicios de almacenamiento en la nube gratuitos como Google Code o GitHub, por citar algunos. De esa forma ROS ya dispone de soporte nativo para un gran número de hardware como Nao1, Lego Mindstorms2, las aspiradoras Roomba3, o incluso robots de investigación como Manfred, o el robot PR2.

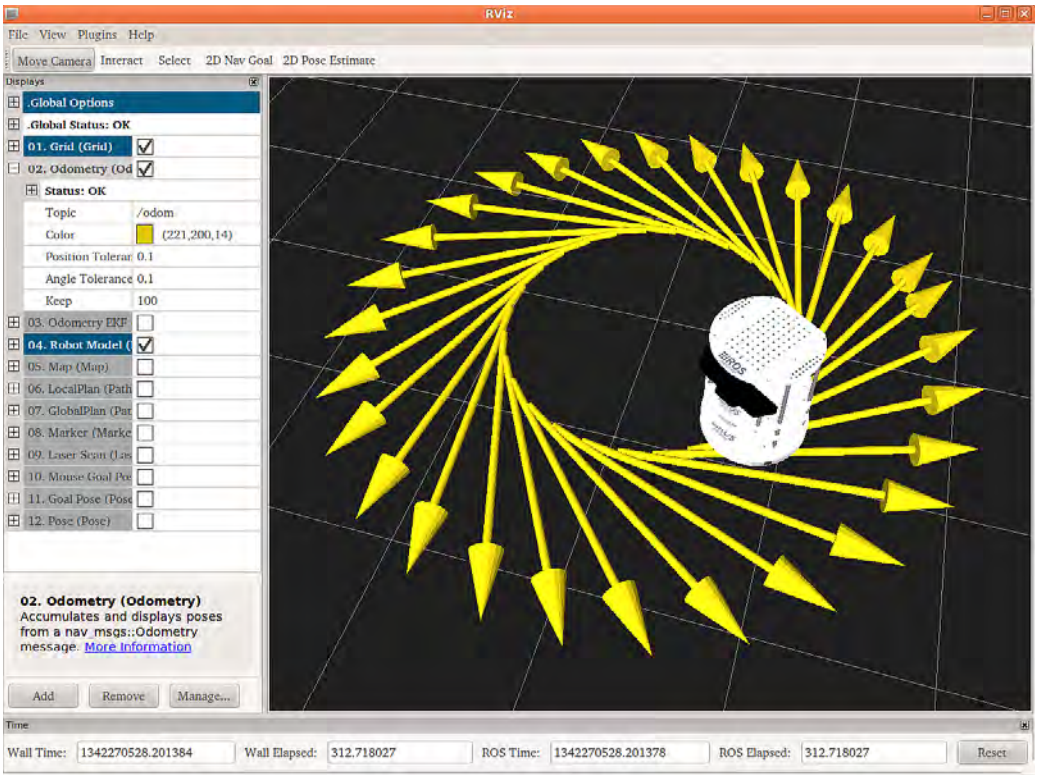

Figura 5.2: Simulador 3D mostrando un robot TurtleBot y un conjunto de vectores que indican su orientación en distintos instantes de tiempo durante su recorrido.

Ha quedado claro que ROS puede usarse en una gran variedad de hardware robótico; sin

embargo, para hacer uso de este framework no es necesario contar con una plataforma física. ROS incluye software para simular el comportamiento de numerosos robots (sensores, odometría, articulaciones, etc.) que se puede usar para comprobar el funcionamiento de los algoritmos desarrollados antes de usarlos en la vida real.
Se puede usar el software de simulación 3D, llamado Gazebo, o el software de simulación 2D, llamado Stage. Ambos simuladores son muy completos e incluyen tanto información de los sensores como simulaciones de interacciones físicas entre objetos. Ver figura 5.2.

ROS se ejecuta mayoritariamente sobre sistemas operativos tipo UNIX/Linux, principalmente Ubuntu y Mac OS X, aunque también existen versiones no completas para otras distribuciones como Fedora, Gentoo, FreeBSD e incluso otros sistemas operativos como Windows. Sin embargo, la manera más sencilla de comenzar a trabajar con ROS es instalándolo en una distribución Ubuntu Linux, ya que ésta es el sistema operativo oficialmente soportado por Willow Garage desde sus comienzos.

Las versiones liberadas del framework ROS liberadas hasta la fecha actual son:

1. ROS Box Turtle (Marzo, 2010)
2. ROS C Turtle (Agosto, 2010)
3. ROS Diamondback (Marzo, 2011)
4. ROS Electric (Agosto, 2011)
5. ROS Fuerte (Abril, 2012)
6. ROS Groovy (Diciembre, 2012)
7. ROS Hydro Medusa (Septiembre, 2013)

5.1. Conceptos generales

Paquete (Package)

Un paquete es el nivel más bajo de organización del software en ROS. Un paquete puede estar constituido por cualquier cosa, librerías de código, herramientas, ejecutables, ficheros de configuración de los ejecutables, etc., todos ellos compartiendo un objetivo común.
Todo el código en ROS debe pertenecer a algún paquete, incluso el software desarrollado por los

usuarios deben pertenecer a su propio paquete. El objetivo de los paquetes es la agrupar recursos que luego pueden ser fácilmente reutilizados en otros proyectos.

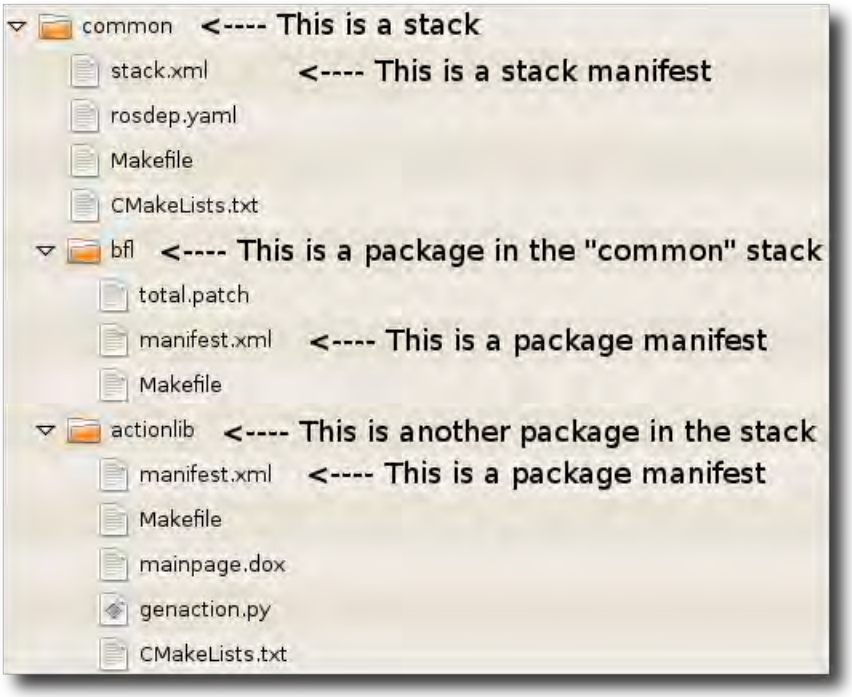

Figura 5.3: Sistema de ficheros de ROS.

Manifiesto (Manifest)

Se trata de un fichero XML que proporciona información sobre un paquete, por ejemplo, tipo de licencia, autores del software, versión del software, distribución de S.O en la que se ha probado, las dependencias de este software con otro software de ROS, etc.

```
<package>
     <description>
          Este paquete se usa para conocer la ubicación (posición y orientación)
del robot con respecto al
          sistema de referencia local que posee en el momento de ser encendido.
     </description>
     <author>jfrascon</author>
     <license>BSD</license>
     <depend package="geometry_msgs"/>
     <depend package="nav_msgs"/>
     <depend package="roscpp"/>
     <depend package="std_msgs" />
     <depend package="tf"/>
     <depend package="pmac"/>
</package>
```

Figura 5.4: Fichero `manifest.xml` del paquete `odometria` desarrollado para el robot Manfred.

Pila (Stack)

Una pila es una colección de paquetes que están relacionados porque desempeñan actividades que comparten un objetivo. Mientras que el objetivo de los paquetes es facilitar la reutilización del código que contienen, el objetivo de las pilas es simplificar el proceso de distribución del software. Por último indicar que ROS no permite el anudamiento de pilas, es decir, no se puede anidar una pila dentro de otra pila.

```
<stack>
  <description brief="common code for personal robots">
    A set of code and messages that are widely useful to all robots. Things
    like generic robot messages (i.e., kinematics, transforms), a generic
    transform library (tf), laser-scan utilities, etc.
  </description>
  <author>Maintained by Tully Foote</author>
  <license>BSD</license>
  <review status="unreviewed" notes=""/>
  <url>http://pr.willowgarage.com/wiki/common</url>
  <depend stack="ros"/>
</stack>
```

Figura 5.5: Ejemplo de fichero `stack.xml`.

Para que el sistema ROS pueda encontrar todos los recursos que le hagan falta durante se

ejecución se debe especificar en la variable de entorno `ROS_PACKAGE_PATH` la lista de directorios (ruta completa desde el directorio raíz) a partir de los cuales cuelgan las pilas y/o paquetes de interés. Es aconsejable que las pilas y/o paquetes desarrollados por los usuarios están separados de las pilas y/o paquetes instalados por ROS.

Manifiesto de pila (Stack manifest)

Se trata del mismo concepto que el manifiesto de paquete, pero aplicado a la pila. Es fácil distinguir un paquete de una pila, ya que el paquete es un directorio que contiene un fichero `manifest.xml`, en cambio una pila es una directorio con un fichero `stack.xml`.

Nodo

Uno de los objetivos básicos de ROS es permitir a la comunidad de desarrolladores diseñar programas que se ejecutan como procesos independientes, denominados *nodos*, todos al mismo tiempo y que se intercambian información. Por ejemplo, un nodo obtiene información de un telémetro láser, otro nodo controla los motores de las ruedas, otro nodo localiza al robot en el entorno, otro nodo realiza la planificación de la trayectoria hasta un destino, otro nodo proporciona una vista del entorno en el que se mueve el robot, etc. El uso de nodos independientes que realizan tareas hace que el sistema ROS sea más sencillo de implementar y más tolerante a fallos que si de un sistema monolítico se tratara. El uso de nodos independientes permite que aunque uno o varios nodos fallen el sistema no tiene que abortar por completo su ejecución, cosa que seguramente sucedería en caso de ser un sistema monolítico.

El nodo más importante de ROS que permite la comunicación de los demás nodos se denomina `ROS master`, o simplemente, nodo maestro. Este nodo se arranca fácilmente desde un terminal con la expresión `roscore` (sin argumentos). Una vez arrancado el nodo maestro se mantendrá en funcionamiento, gestionando todo el sistema ROS, hasta que el usuario decida acabar con él de manera explícita pulsando la combinación de teclas <Ctrl-C>. Una metodología de trabajo adecuada consiste en arrancar este nodo en un terminal y arrancar el resto de nodos en otros terminales.

ROS dispone de un sistema de nombrado de recursos (nodos, parámetros, servicios) que permite identificar de manera unívoca a cual de ellos se hace referencia. En el caso de los nodos no hay que confundir el nombre que se le da al fichero ejecutable con el nombre con el que

ese nodo se registra en el nodo maestro. Así un ejecutable puede llamarse `teleop.bin` y al ser arrancado registrarse en el nodo maestro como `/nodo_teleop`.

ROS dispone del concepto de *espacio de nombres*, con el mismo uso que el lenguaje C++. Un espacio de nombre es un grupo lógico de elementos identificados por un nombre común. Los espacios de nombres permiten que existan elementos del sistema con el mismo nombre siempre y cuando pertenezcan a espacios de nombres distintos. Al igual que en C++ los espacios de nombres se pueden anidar en tantos niveles como se desee. Siguiendo con el ejemplo anterior se puede disponer de dos instancias globales del ejecutable `teleop.bin`, compartiendo el mismo nombre, siempre que pertenezcan a espacios de nombres diferentes, `/lab1/nodo_teleop` y `/lab2/nodo_teleop`

Cualquier nombre de recurso que se especifique de forma relativa será expandido añadiéndole de forma implícita el nombre del espacio de nombres indicado por defecto para el sistema. Así si por ejemplo el espacio de nombre por defecto para el sistema es `/manfred`, un nombre de recurso relativo, por ejemplo para un nodo es `nodo_teleop`, dicho nombre será registrado en el nodo maestro como como `/manfred/nodo_teleop`. Cualquier uso del nombre `nodo_teleop` dentro del sistema será expandido a `/manfred/nodo_teleop`. Por poner otro ejemplo de nombre relativo de recurso con espacios de nombres anidados, considere el siguiente nodo `/lab1/nodo_teleop`. Si el espacio de nombres por defecto elegido para el sistema robótico es `/manfred`, el nombre expandido del recurso es `/manfred/lab1/nodo_teleop`.

Es importante que cada nodo se registre en el nodo maestro con un nombre único, de lo contrario el nodo maestro impedirá que dicho nodo continué ejecutándose, finalizándolo inmediatamente.

El espacio de nombres por defecto para el sistema robótico se indica a través de la variable de entorno `ROS_NAMESPACE`. Si no se especifica espacio de nombres en esta variable se considera por defecto que es /.

Mensaje (Topic)

En ocasiones para que los nodos puedan funcionar adecuadamente necesitan información de entrada que es proporcionada por otros nodos. Esta información se intercambia entre nodos a través de mensajes. Los mensajes son estructuras de datos que especifican explícitamente que tipo de información se intercambia entre dos nodos. Los mensajes están constituidos por campos

de tipos primitivos, como por ejemplo `int`, `float`, `boolean`, etc. Los mensajes pueden incluir estructuras arbitrariamente anidadas y matrices.

Al intercambio de mensajes entre dos nodos se le asigna un nombre identificativo denominado `topic`[3]. Es recomendable que el `topic` de un mensaje sea escogido con cuidado, de manera que describa brevemente la información que intercambian los nodos en contacto.

Para que dos nodos pueden intercambiar información el nodo productor de dicha información debe registrarse en el nodo maestro como un *publicador* de un tipo concreto de mensajes, bajo un determinado nombre de topic. Por ejemplo supongamos un nodo publicador de mensajes de tipo imagen, ver figura 5.6, con el asunto (`topic`) `Imagen_cámara_frontal`.

```
# This message contains an uncompressed image
# (0, 0) is at top-left corner of image
#

Header header        # Header timestamp should be acquisition time of image
                     # Header frame_id should be optical frame of camera
                     # origin of frame should be optical center of cameara
                     # +x should point to the right in the image
                     # +y should point down in the image
                     # +z should point into to plane of the image
                     # If the frame_id here and the frame_id of the CameraInfo
                     # message associated with the image conflict
                     # the behavior is undefined

uint32 height         # image height, that is, number of rows
uint32 width          # image width, that is, number of columns

# The legal values for encoding are in file src/image_encodings.cpp
# If you want to standardize a new string format, join
# ros-users@lists.sourceforge.net and send an email proposing a new encoding.

string encoding       # Encoding of pixels -- channel meaning, ordering, size
                      # taken from the list of strings in include/sensor_msgs/image_encodings.h

uint8 is_bigendian    # is this data bigendian?
uint32 step           # Full row length in bytes
uint8[] data          # actual matrix data, size is (step * rows)
```

Figura 5.6: Mensaje del tipo `std_msgs/msg/Image` usado para transmitir información visual.

Un campo que poseen todos los menajes es el `Header`, que está constituido por numerosos subcampos que contienen metadatos a cerca del mensaje como por ejemplo una marca temporal para identificar en qué momento se ha generado el mensaje. Los otros dos campos que constituyen la cabecera son el número de secuencia (`seq`), incrementado automáticamente por el framework ROS, y el identificador del sistema de referencia (`frame_id`) con respecto al que se han tomado los

[3]Por hacer un símil que facilite la compresión de este concepto, el topic es como el campo *Asunto* en los correos electrónicos

datos. Por ejemplo, si un mensaje transporta información de un barrido láser la marca temporal indicaría en que momento se tomó el barrido y el `frame_id` indicaría desde que sistema de referencia se ha tomado dicho barrido.

```
#Standard metadata for higher-level flow data types
#sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.secs: seconds (stamp_secs) since epoch
# * stamp.nsecs: nanoseconds since stamp_secs
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
# 0: no frame
# 1: global frame
string frame_id
```

Figura 5.7: Estructura de datos `Header` incluida en cada mensaje que se transfiere entre dos nodos.

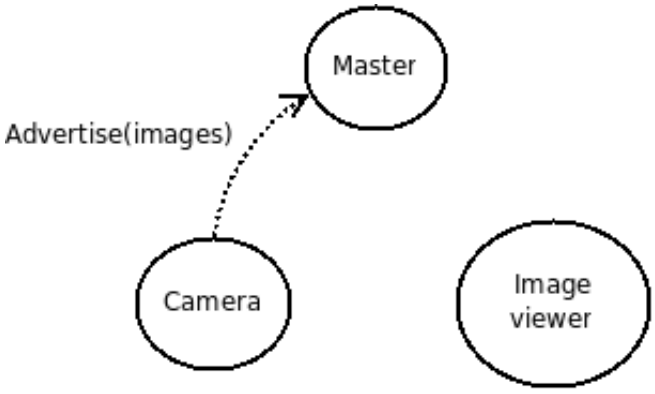

Figura 5.8: Nodo indicando al nodo maestro su intención en difundir mensajes de tipo imagen.

El nodo consumidor de un tipo concreto de información también debe registrar en el nodo maestro su interés por esa clase de información vinculada a un determinado topic. Cabe destacar que en un sistema ROS pueden existir varias fuentes productoras del mismo tipo de información, publicadas en diferentes topics, de modo que un suscritor de un tipo de información para un determinado topic, no recibirá el resto de informaciones del mismo tipo, pues éstas informaciones

son publicadas con topics distintos a los que no está suscrito.

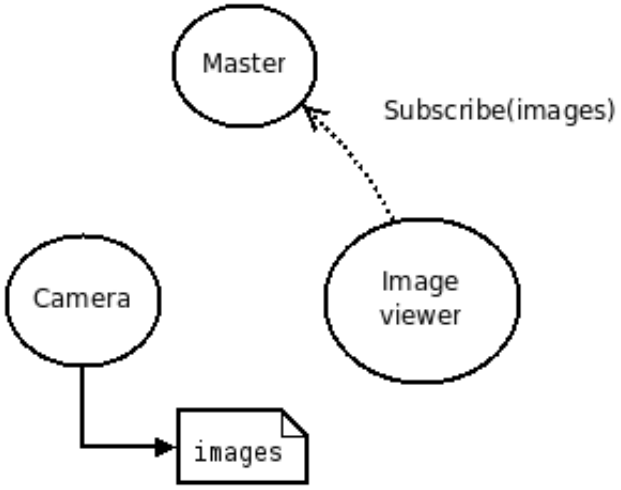

Figura 5.9: Nodo indicando al nodo maestro su interés por recibir mensajes de tipo imagen.

Una vez que el nodo maestro tiene registrada la información de los publicadores y suscriptores de información, éste dispone lo necesario para que ambos nodos puedan intercambiarse información tan pronto como ésta es obtenida. De este modo se desacopla la fuente de producción de información de la fuente consumidora de la misma.

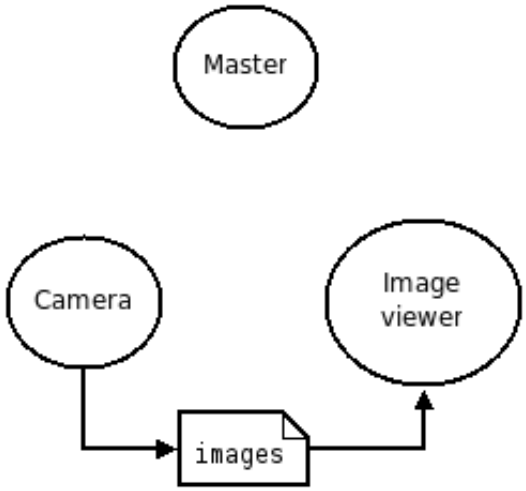

Figura 5.10: Nodo indicando al nodo maestro su interés por recibir mensajes de tipo imagen.

Cabe destacar que un nodo puede suscribirse a varios topics, no exclusivamente a uno.

Servicio (Service)

En los mensajes la información se desplaza en un único sentido, del productor al consumidor. En aquellas situaciones donde sea necesario que un nodo le solicite a otro que haga una tarea por él y le devuelva los resultados de la misma se necesita un servicio. La comunicación en estos casos es bidireccional. Un servicio es una estructura de datos con la cual un nodo le indica a otro que actividad debe realizar para él y si éste último debe devolverle resultados y con que formato.

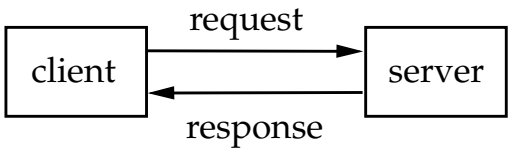

Figura 5.11: Mecanismo de comunicación mediante servicios en ROS.

De una forma similar o lo que sucedía con los publicadores/suscriptores de mensajes, los nodos que ofrecen y demandan servicios deben registrar su actividad en el nodo maestro. Así lo nodos que realizan servicios registran en el nodo maestro que función pueden realizar y la vinculan a un nombre. Los nodos que demandan servicios simplemente registran qué servicio desean usar.

Tanto los mensajes como los servicios disponen de una suma de verificación que se comprueba en los nodos extremos para verificar que la información no se ha corrompido durante su intercambio.

Los mensajes y los servicios entre nodos son transportados mediante sockets TCP o sockets UDP. Durante el registro de un nodo publicador, suscriptor, servidor o solicitante de un servicio, estos negocian como debe transportarse la información.

Parámetro (Parameter)

Un servidor de parámetros es un diccionario compartido que puede almacenar elementos de categorías distintas y que puede ser accedido a través de la red. Los nodos usan el servidor de parámetros para almacenar y obtener parámetros en tiempo de ejecución. El servidor de parámetros es parte del nodo maestro, así que está disponible desde el mismo momento que se arranca éste. El servidor de parámetros es adecuado para almacenar información estática que deba ser visible globalmente, por ejemplo, información de configuración del robot. Así, un nodo

puede inspeccionar cual es la configuración del robot y si fuera necesario cambiarla. Los valores que se pueden almacenar en el servidor de parámetros deben pertenecer a uno de los siguientes tipos:

- `int`.
- `double`.
- `boolean`.
- `string`.
- `iso8601 date`.
- `list`.
- `base64-encoded binary data`.

El sistema de nombrado de los parámetros hace uso de los espacios de nombres, al igual que los mensajes y los servicios, para evitar colisiones de nombres. Por ejemplo:

```
/camera/left/name: leftcamera
/camera/left/exposure: 1
/camera/left/name: rightcamera
/camera/left/exposure: 1.1
```

Es posible acceder al valor de los parámetros de maneras distintas. Por ejemplo el parámetro `/camera/left/name` tiene el valor `leftcamera`, sin embargo, también se puede acceder al valor de `/camera/left`, que es el diccionario:

```
name: leftcamera
exposure: 1
```

También es posible acceder al valor de `/camera`, cuyo valor es un diccionario de diccionarios:

```
left: {name: leftcamera, exposure: 1}
right: {name: leftcamera, exposure: 1}
```

5.2. Paquetes relevantes del framework ROS usados

5.2.1. El paquete `tf`

En el framework ROS se define un árbol de transformación como el conjunto de transformaciones (desplazamientos y rotaciones) que hay que aplicar a unos datos para representarlos en un sistema de referencia diferente de aquel en el que han sido obtenidos. ROS proporciona una forma sencilla de expresar la transformación existente entre dos sistemas de referencia a través del paquete `tf`. Para ilustrar este concepto mejor se considerará un ejemplo en el que se dispone de un simple sistema robótico conformado por una base móvil con un escáner láser sobre ella. En ese robot se definen dos sistemas de referencia. El primero situado en el centro de la base móvil recibe el nombre `base_link`. El segundo sistema de referencia se haya situado en el centro del sensor láser y recibe el nombre de `base_laser`.

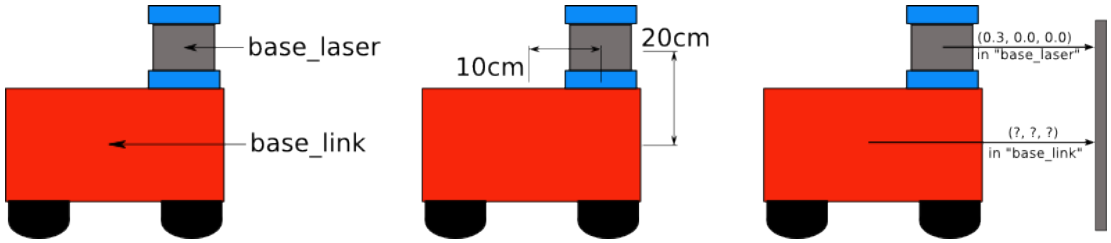

Figura 5.12: Relación entre diferentes sistemas de referencia.

A continuación se asume que se dispone de ciertos datos proporcionados por el sensor láser. Concretamente se dispone de la distancia desde el centro del láser a un obstáculo situado frente al robot. Ahora supongamos que queremos usar estos datos para ayudar a la base móvil a evitar obstáculos en el entorno. Para llevar a cabo esta tarea es preciso transformar las coordenadas referidas al sistema de referencia `base_laser` en coordenadas referidas al sistema de referencia `base_link`.

Para llevar a cabo tal transformación en primer lugar hay que averiguar la relación que existe en los sistemas de referencia `base_laser` y `base_link`. Supongamos que conocemos esta relación y que definimos el sistema de referencia `base_link` como sistema de referencia principal, mientras que el sistema de referencia `base_laser` es un sistema de referencia secundario. Tal y como se aprecia en la figura 5.12 el sensor láser se haya montado sobre la base móvil, 10 cm por delante del centro de la base y 20 cm por encima del centro de la base. Estos desplazamientos de traslación

son los que relacionan ambos sistemas de referencia.

$$\left(\Delta x = 0'1\,\mathrm{m},\ \Delta y = 0'0\,\mathrm{m},\ \Delta z = 0'2\,\mathrm{m}\right)^T \tag{5.1}$$

Por tanto si a las coordenadas de un punto respecto del sistema de referencia `base_laser` se le aplica el vector de traslación $(\Delta x,\ \Delta y,\ \Delta z)^T$ se obtienen las coordenadas de dicho punto con respecto al sistema de referencia `base_link`. Por otro lado si a las coordenadas de un punto respecto del sistema de referencia `base_link` se le aplica el vector de traslación $(-\Delta x,\ \Delta y,\ -\Delta z)^T$ se obtienen las coordenadas de dicho punto con respecto al sistema de referencia `base_laser`.

$$\begin{bmatrix} {}^{b.link}x \\ {}^{b.link}y \\ {}^{b.link}z \end{bmatrix} = \begin{bmatrix} {}^{b.laser}x \\ {}^{b.laser}y \\ {}^{b.laser}z \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{5.2}$$

Está relación entre los sistemas de referencia podría ser implementada por el desarrollado en el código fuente, aplicándola en la sentido adecuado dependiendo del tipo de transformación que se quiera aplicar. Sin embargo este enfoque se vuelve inmanejable a medida que el número de sistemas de referencia aumenta en el sistema robótico. Afortunadamente este trabajo no tiene que ser implementado por el desarrollador. En su lugar se le puede indicar al paquete `tf` cual es la relación entre los sistemas de referencia y dicho paquete se encarga de aplicar las transformaciones entre diferentes sistemas de coordenadas por nosotros.

Para definir la relación entre dos sistemas de referencia en el paquete `tf` se necesita crear un *árbol de transformaciones*. Un árbol de transformaciones es un grado dirigido acíclico en el que cada nodo representa un sistema de referencia y cada enlace representa la transformación necesaria para cambiar las coordenadas tomadas en un sistema de referencia al otro sistema de referencia y viceversa. Sólo hay un enlace que une un nodo padre con un nodo hijo. Los enlaces están dirigidos desde el nodo padre al nodo hijo. La transformación debe definirse de modo que cambie las coordenadas desde el sistema de referencia hijo al sistema de referencia padre. Ver figura 5.13.

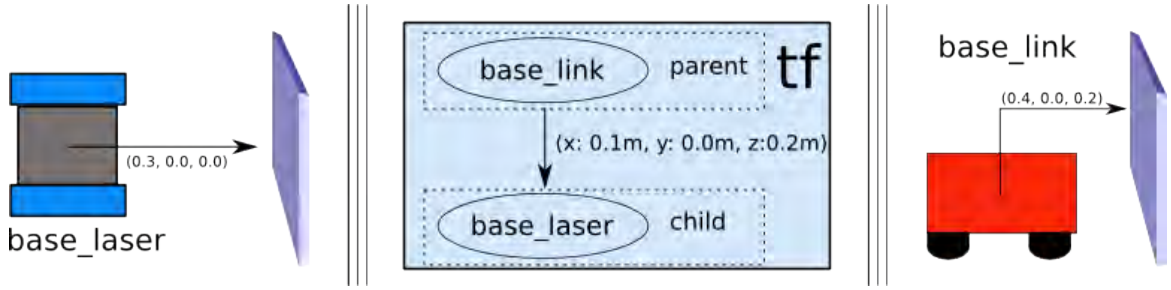

Figura 5.13: Árbol de transformaciones definido haciendo uso del paquete `tf`.

Siguiendo con el ejemplo de la base móvil, el árbol de transformación necesario consta de dos nodos, el nodo padre `base_link` y el nodo hijo `base_link`. Se suele escoger como nodo padre al sistema de referencia `base_link` ya que ésta suele ser el lugar común en el que se sitúan numerosos sensores de diversas clases. Habiendo determinado el nodo padre y el nodo hijo tan sólo resta definir la transformación entre ambos nodos. Siguiendo la regla descrita anteriormente la transformación necesaria es:

$$(\Delta x,\, \Delta y,\, \Delta z)^T$$

Una vez definido el árbol de transformaciones del sistema robótico del ejemplo convertir las coordenadas de un punto respecto del sistema de referencia `base_laser` al sistema de referencia `base_link` es tan sencillo como hacer una llamada a una función de la librería `tf` indicando el sistema de referencia destino de la transformación. Es necesario mencionar que también se le puede pedir a la librería `tf` que transforme las coordenadas de un punto expresadas en el marco de referencia `base_link` a coordenadas expresadas en el sistema de referencia `base_laser`. La biblioteca `tf` se puede emplear en ambos sentidos. Ahora el robot móvil puede usar la información proporcionada por el sensor láser para navegar de forma segura alrededor de los obstáculos situados en el entorno.

5.2.2. El modelo visual del robot mediante un fichero `urdf`

El árbol de transformaciones del robot Manfred se crea con el paquete `urdf` del framework `ROS`. Este árbol de transformaciones permitirá a la biblioteca `tf` obtener las coordenadas de un punto con respecto a un marco de referencia diferente de aquel en el que fueron obtenidas.

Un aspecto muy importante cuando se trabaja en el campo de la robótica es contar con un modelo de la plataforma donde se puedan probar los algoritmos desarrollados. Este modelo debe

reproducir con la mayor exactitud posible las características el sistema robótico. Habitualmente también es necesario modelar el entorno en el que opera el robot. Cuanto más realistas sean ambos modelos más se aproximan los resultados de la simulación a los resultados obtenidos en la realidad.

En ROS se puede construir un árbol de transformaciones de cualquier sistema robótico, por complicado que éste sea, a partir de un fichero XML que sigue un formato propietario denominado *Unified Robot Description Format*.

Un fichero `urdf` puede estar constituido por numerosos elementos, de los cuales destacan dos, por su importancia ya que siempre han de estar presentes, los eslabones (etiqueta xml `<link>`) y las articulaciones (etiqueta xml `<joint>`). Debido a su importancia, y a que el resto de elementos son opcionales, sólo de describirán estos dos elementos.

La palabra eslabón es muy genérica en este contexto. Básicamente significa cualquier pieza del robot que posea un marco de referencia que se quiera incluir en el árbol de transformaciones. Por tanto un eslabón puede ser una pieza mecánica, por ejemplo una rueda, el torso del robot, cada uno de los fragmentos de un brazo manipulador, una pinza de agarre, un sensor de telemetría láser, un cámara de visión 3D, etc.

Por articulación se entiende el punto de unión de dos eslabones contiguos por el que uno de ellos puede moverse con respecto al otro.

Tanto los eslabones como las articulaciones poseen multitud de propiedades que se pueden editar, como por ejemplo la forma y tamaño de los eslabones, la ubicación del marco de referencia del eslabón dentro del mismo, el tipo de movimiento que pueden realizan los diferentes eslabones alrededor de las articulaciones, etc.

A continuación se van a explicar los elementos más importantes que forman parte de un fichero `urdf` mediante un ejemplo en el que se construye una versión simplificada del robot Manfred. En muchas ocasiones construir un modelo visual simplificado de un robot a partir de figuras geométricas sencillas es más que suficiente para simular el comportamiento del sistema robótico. En estos casos es imprescindible que las figuras geométricas representen cajas de contorno de los componentes verdaderos del robot respetando sus dimensiones exactas.

En las figuras 5.14, 5.15 y 5.18 se pueden apreciar distintas perspectivas del modelo visual que se usará como ejemplo para describir las distintas partes que componen un fichero `urdf`.

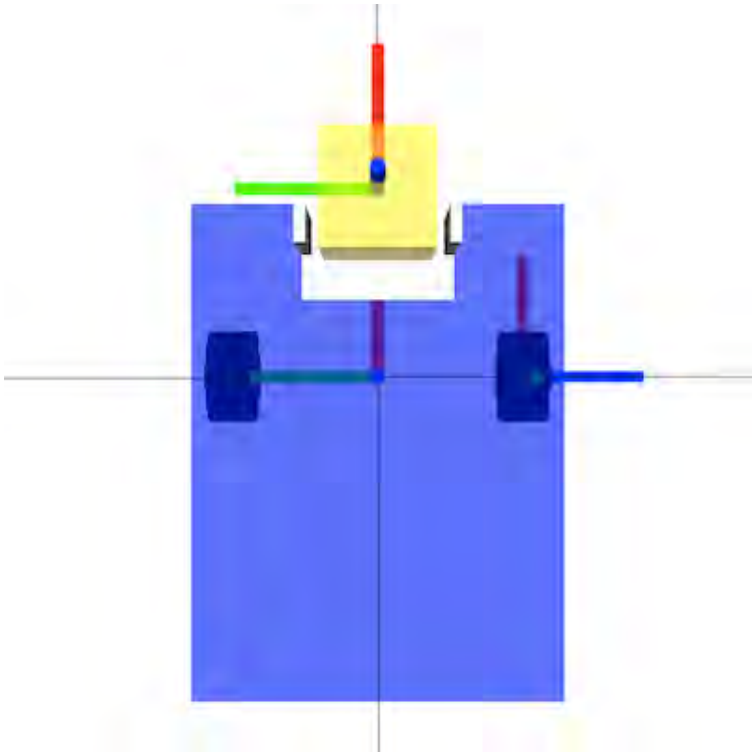

Figura 5.14: Vista superior del modelo visual simplificado de Manfred en ROS.

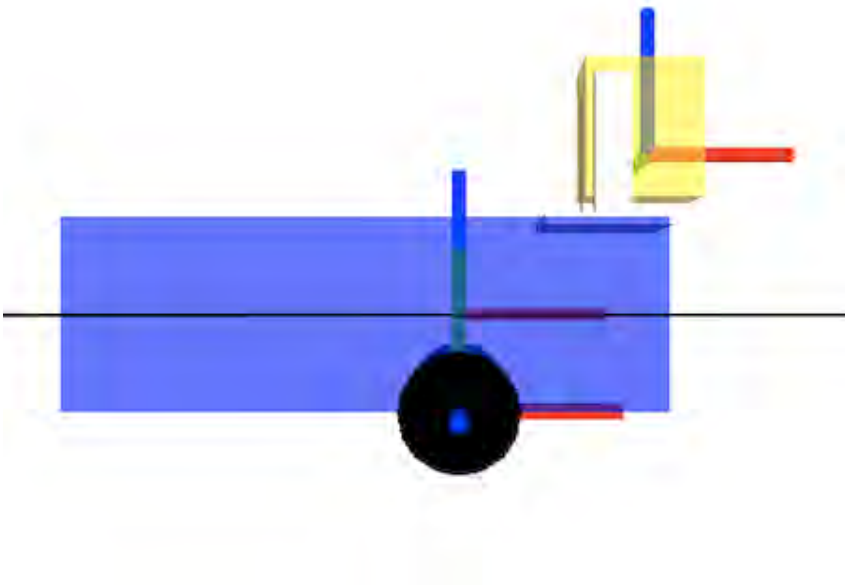

Figura 5.15: Vista lateral derecha del modelo visual simplificado de Manfred en ROS.

Cuando se crea un fichero `urdf` no existe un orden específico en el que definir los eslabones y articulaciones, sin embargo, la tendencia observada en todos los ficheros `urdf` consultados es que o bien se definen primeros todos los eslabones que constituyen el robot y luego todas las articulaciones o bien se describen dos eslabones consecutivos, a continuación la articulación que los une y el resto del fichero sigue el mismo procedimiento. El segundo procedimiento suele ser más cómodo para trabajar, requiere de menos desplazamientos por el fichero buscando los elementos de interés.

A modo de ejemplo se detalla el fragmento de fichero urdf que describe la base móvil de Manfred.

```
<link name="link_base">
    <visual>
        <geometry>
            <box size="0.730 0.550 0.235" />
        </geometry>
        <material name="Azul">
            <color rgba="0.000 0.000 1.000 1"/>
        </material>
        <origin xyz="-0.112 0 0" rpy="0 0 0" />
    </visual>
</link>
```

Como ya se ha mencionado la etiqueta `<link>` se usa para definir un eslabón del robot. El atributo `name` indica el nombre con el que se identifica a ese eslabón en el sistema robótico. Dentro de la etiqueta `<link>` aparece la etiqueta `<visual>`, la cual describe, como su nombre indica, el aspecto visual del eslabón. La etiqueta `<geometry>` se usa para definir la forma que tiene el eslabón. En este caso concreto la base tiene forma de prisma rectangular, indicado por la etiqueta `<box>`. Las dimensiones del prisma rectangular se indican en el atributo `size`, en el siguiente orden, dimensión en el eje `x`, dimensión en el eje `y` y finalmente dimensión en el eje `z`, todas ellas expresadas en metros.

Las formas geométricas básicas adicionales que se pueden usar dentro la etiqueta `<geometry>` son cilindro (`<cylinder>`), caracterizada por su atributo radio y altura y esfera (`<sphere>`), caracterizada por su radio.

También es posible usar formas complicadas, definidas a base de mallas. Estas formas suelen realizarse con software específico de modelización, por ejemplo el programa open source *MeshLab*. Para importar ficheros de mallas se usa la etiqueta `<mesh>`, indicando en el atributo `<filename>` el fichero de mallas, ruta incluida. Los formatos de ficheros de mallas soportados son `dae` y `stl`. La ventaja del fichero `dae` es que incorpora información de color. Las mallas pueden ser escaladas para acomodarse al tamaño necesario.

En el ejemplo siguiente podemos observar al robot R2D2 construido con formas geométricas básicas excepto su pinza, importada de un fichero `dae` en el que se define a base de mallas. Ver figura 5.16.

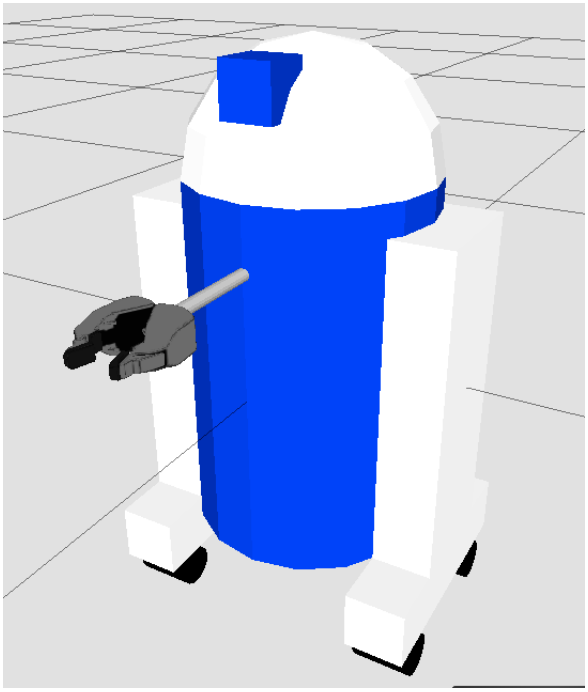

Figura 5.16: Vista general del robot R2D2 construido con formas geométricas básicas excepto la pinza, construida con mallas.

```
<link name="left_gripper">
  <visual>
    <origin rpy="0.0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://pr2_description/meshes/gripper_v0/l_finger.dae"/>
    </geometry>
```

```
  </visual>
</link>
```

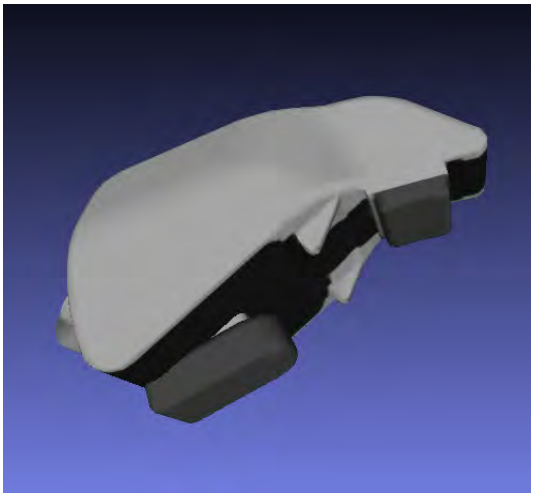

Figura 5.17: Detalle del dedo izquierdo de la pinza construido con mallas.

La etiqueta `<material>` sirve para especificar el color del eslabón. El color puede ser incorporado al eslabón bien a través de la etiqueta `<color>`, compuesta por cuatro números en el rango [0, 1] que representan las coordenadas r/g/b/alpha, o bien por la etiqueta `<texture>`, que puede importar la textura desde un fichero imagen, por ejemplo, jpg, png, etc.

Por defecto el marco de referencia del eslabón se haya situado en el centro geométrico del mismo. En ocasiones el marco de referencia del eslabón tiene que estar en otra ubicación diferente del centro geométrico del eslabón. Para ello se cuenta con la etiqueta `<origin>`. Esta etiqueta sirve para indicar la ubicación tridimensional del centro geométrico del robot con respecto al marco de referencia del eslabón por medio de los atributos `xyz` y `rpy`. Es decir, el centro geométrico del eslabón, acompañado del resto del eslabón, se desplaza a una nueva ubicación con respecto al marco de referencia. El resultado visual es que el marco de referencia del robot se haya en una ubicación distinta del centro geométrico. En el ejemplo propuesto el centro geométrico de la base se encuentra en las coordenadas ($x = -0'112$ m, $y = 0$ m, $z = 0$ m, $r = 0$ rad, $p = 0$ rad, $y = 0$ rad) de su marco de referencia. Ver figuras 5.14 y 5.15. El marco de referencia de la base se ha puesto en esa ubicación para que se encuentre sobre el punto medio del eje imaginario que une los centros de las ruedas motoras de la base diferencial. Ver figura 5.15. Cualquier punto que se encuentre en la vertical del centro geométrico de este eje imaginario se puede usar como represente del robot

en los algoritmos de localización goblal y local. Y de todos los posibles puntos sobre esa vertical se ha considerado que el más optimo es el que se encuentra en la altura media de la base, ya que ésta es el lugar donde numerosos sensores del robot son emplazados y por tanto un lugar de referencia. Por tanto, este punto de la base es el elegido para situar su marco de referencia.

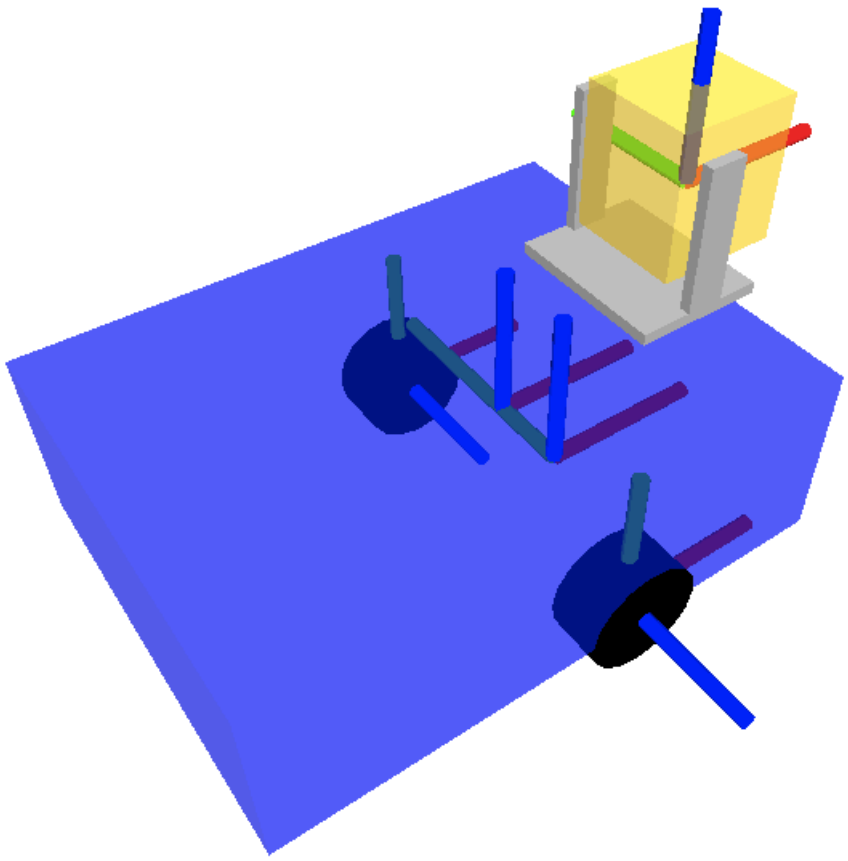

Figura 5.18: Vista general del modelo visual simplificado de Manfred en ROS.

Por último cabe destacar que si lo que se desea es ubicar un marco de referencia en un punto concreto del robot, sin estar asociado a una figura geométrica básica o cuerpo construido con mallas se debe usar la etiqueta `<link>` que no posea una etiqueta `<visual>`. De este modo se van a poder referir coordenadas con respecto a este marco de referencia. Este modo de proceder puede usarse cuando se quiere ubicar un segundo marco de referencia dentro de un eslabón que ya posee un marco de referencia, como ocurre en el caso de la base móvil del robot. En la figura 5.18 puede observarse un marco de referencia situado a la derecha del marco de referencia del eslabón base, a su misma altura. El punto en el que sitúa este nuevo marco de referencia recibe

el nombre de `link_base_desp`. Este nuevo marco de referencia es necesario debido a que el robot Manfred no es simétrico en su plano sagital. Manfred posee un brazo manipulador de 7 grados de libertad situado en su lateral derecho. Este brazo manipulador sobresale 10 cm de la base. Esta asimetría provoca que cuando el robot sigue una trayectoria no es el marco de referencia `link_base` el que tiene que pasar por encima de los puntos de control situados en la trayectoria, si no que es el marco de referencia desplazado, `link_base_desp`, el que tiene que pasar sobre ellos, pues es este sistema de referencia el que se ubica en el punto medio de la anchura del sistema robótico, sobre el eje alrededor del cual rotan las ruedas.

```
<link name="link_base_desp" />

<joint name="joint_base_base_desp" type="fixed">
    <parent link="link_base" />
    <child link="link_base_desp" />
    <origin xyz="0 -0.050 0" rpy="0 0 0" />
</joint>
```

A continuación se va a describir como se define una articulación. Tal y como se dijo previamente se puede considerar una articulación como el punto de contacto entre dos eslabones consecutivos por el que uno de ellos se mueve relativo al otro. Uno de los eslabones ejerce de elemento de referencia, denominado *eslabón primario*. Este eslabón no se mueve en la articulación. El eslabón restante, que se ubica con respecto al primero, se le denomina *eslabón secundario*. Este es el eslabón que se mueve en la articulación. A su vez el eslabón secundario puede comportarse como eslabón primario con respecto a otro eslabón, y así sucesivamente se va formando un árbol de eslabones ordenados por importancia.

Siguiendo con el ejemplo propuesto se definirá la articulación que une la base móvil con la rueda motriz derecha.

```
<joint name="joint_base_rueda_derecha" type="continuous">
    <parent link="link_base"/>
    <child link="link_rueda_derecha"/>
    <origin xyz="0 -0.238 -0.119" rpy="1.571 0 0" />
    <axis xyz="0 0 1"/>
</joint>
```

La etiqueta que permite definir una articulación es `<joint>`. Cada articulación debe recibir un nombre que la identifique en el sistema robótico e indicar cual es el movimiento que puede realizar el eslabón secundario en ella. Para poder entender adecuadamente el uso del atributo `type` antes es necesario explicar el significado las etiquetas `<parent>` y `<child>`.

Las etiquetas `<parent>` y `<child>` se usan para identificar al eslabón primario y secundario respectivamente. En nuestro ejemplo el eslabón de referencia es `link_base` y el eslabón que se ubica con respecto a éste es `link_rueda_derecha`. El punto de contacto entre el eslabón primario y secundario viene determinado por la ubicación del marco de referencia del eslabón secundario relativo al marco de referencia del eslabón primario. Para ubicar el eslabón secundario con respecto al primario se usan los atributos `<xyz>` y `<rpy>` la etiqueta `<origin>`. En el punto de contacto de ambos eslabones, coincidente con el marco de referencia del eslabón secundario, se encuentra el marco de referencia de la articulación. En el ejemplo propuesto el marco de referencia de la articulación y de la rueda se ubican en las coordenadas ($x = 0\,\text{m}$, $y = -0'238\,\text{m}$, $z = -0'119\,\text{m}$, $r = 1'571\,\text{rad}$, $p = 0\,\text{rad}$, $y = 0\,\text{rad}$) con respecto al marco de referencia de la base. Cuando se construye un eslabón con forma de cilindro, como en el caso de la rueda, éste siempre tiene su cara circular paralela al suelo sobre el que descansa el robot. Para que la cara circular de la rueda sea perpendicular al plano del suelo es necesario que el marco de referencia de la rueda, y por tanto también el de la articulación, rote $\pi/2\,\text{rad}$ en sentido antihorario con respecto al eje `x` del marco de referencia de la base, de ahí el valor de la coordenada `r`$= 1'571$.

Los movimientos que el eslabón secundario y su marco de referencia pueden realizar en torno al marco de referencia de la articulación (atributo `<type>`) son:

1. Movimiento de revolución limitado: Indica que el eslabón secundario rota alrededor de uno de los ejes del marco de referencia de la articulación, entre un límite angular superior y un límite angular inferior.

2. Movimiento de revolución continuo: Indica que el eslabón secundario rota alrededor de uno de los ejes del marco de referencia de la articulación de manera continua.

3. Movimiento prismático: Indica que el eslabón secundario se desplaza a lo largo de uno de los ejes del marco de referencia de la articulación, entre un límite superior y un límite inferior.

4. Movimiento planar: Indica que el eslabón secundario se desplaza en un plano perpendicular a uno de los ejes del marco de referencia de la articulación, entre un límite superior y otro inferior para cada coordenada que defina el plano de movimiento.

5. Movimiento flotante: Indica que el eslabón secundario puede moverse de manera tridimensional en el espacio, posición y orientación, relativo al marco de referencia de la articulación.

6. Sin movimiento: Indica que el eslabón secundario no puede moverse, está fijo.

Para aquellos casos donde el eslabón secundario realice un movimiento de revolución o un movimiento prismático restringido entre un límite superior y otro inferior se deben especificar estos mediante los atributos `upper` y `lower` de la etiqueta `limit` respectivamente. Si el eslabón secundario puede realizar un movimiento de revolución estos valores se interpretan con la unidad radián. En cambio, si el eslabón secundario puede realizar un movimiento prismático estos valores se interpretan con la unidad metro.

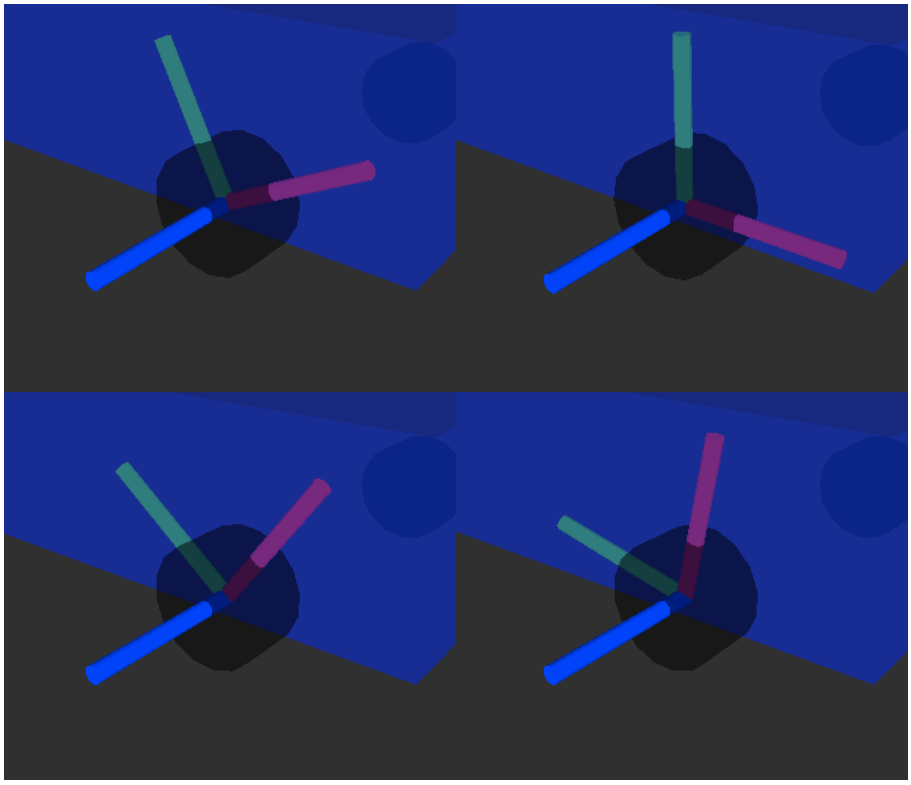

Figura 5.19: Cuadrante 1: 0°. Cuadrante 2: 25°. Cuadrante 3: 50°. Cuadrante 4: 75°

Para indicar el eje del marco de referencia de la articulación alrededor del cual se mueve el esla-

bón secundario y su marco se usa el atributo `xyz` de la etiqueta `<axis>`. En nuestro ejemplo la rueda está definida como un cuerpo que rota de forma continua (`<joint ... type=‘‘continuous’’>`) alrededor del eje `z` del marco de referencia de la articulación, `<axis xyz="0 0 1/>`. Una cosa que se debe tener en cuenta es que tanto el eslabón secundario como su marco de referencia rotan en torno al marco de referencia de la articulación, es decir, el marco de referencia del eslabón secundario no permanece estático. Ver figura 5.19.

Es posible hacer que el eslabón secundario y su marco se muevan respecto a un eje que sea distinto de los ejes del marco de referencia de la articulación. Este eje se haya definido por un vector que pasa por dos puntos del espacio. El primer punto es el origen del marco de referencia de la articulación, cuyas coordenadas son (`x` $= 0$, `y` $= 0$, `z` $= 0$). El segundo punto se especifica en el atributo `xyz` de la etiqueta `<axis>`. Las coordenadas del vector indicadas en el atributo `xyz` deben estar normalizadas.

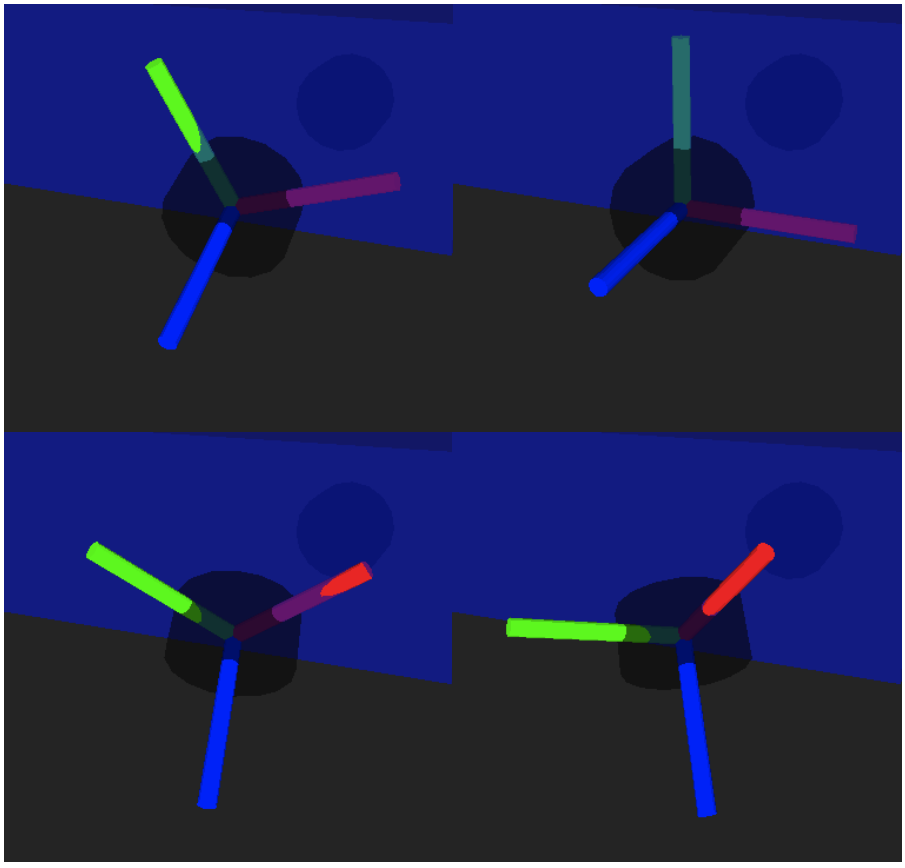

Figura 5.20: Cuadrante 1: 0°. Cuadrante 2: 25°. Cuadrante 3: 50°. Cuadrante 4: 75°

Para ilustrar el concepto se expone un ejemplo. Supongamos que deseamos que la rueda rote, únicamente por motivos ilustrativos, alrededor del eje definido por el vector que pasa por los puntos ($x = 0, y = 0, z = 0$) y ($x = 1, y = 0, z = 1$). Por tanto, ya que la norma del vector anterior es $\sqrt{2}$, las coordenadas normalizadas que deben aparecer en la etiqueta `<axis>` son:

```
<axis xyz="0’707 0 0’707"/>
```

En la figura 5.20 se observan distintas instantáneas que muestran la rotación de la rueda motriz derecha del robot alrededor del eje indicado por la última etiqueta `<axis>`.

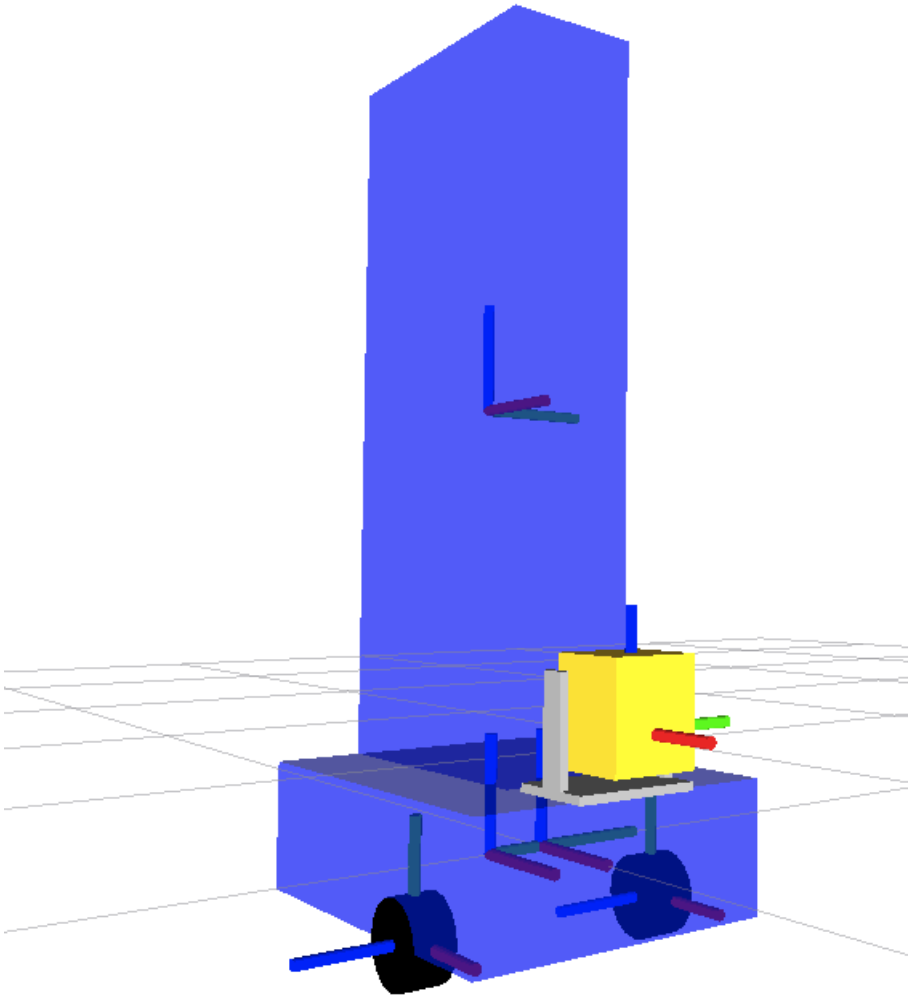

Figura 5.21: Modelo visual de Manfred en ROS.

Capítulo 6

Implementación del algoritmo Pure-Pursuit

En el capítulo 3 se indicó que la expresión principal del algoritmo pure-pursuit sólo posee un parámetro de control, a saber, la distancia desde la posición ocupada por el robot hasta el siguiente punto de control destino elegido en la trayectoria de interés, L_{ah}.

$$R_{sF} = \frac{L_{ah}}{2 \cdot \sin(\theta_{err})} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{2 \cdot \sin\left(\arctan\left(\frac{\Delta y}{\Delta x}\right) - \theta_r\right)} \tag{6.1}$$

Sin embargo en el capítulo 3 no se indico cómo se controla la velocidad de cada rueda para asegurar que el robot sigue la trayectoria de interés. Esta sección tiene por objetivo presentar la implementación del algoritmo pure-pursuit dentro del framework ROS.

En la implementación del algoritmo pure-pursuit se ha fijado un requisito:

El robot debe desplazarse siempre hacia delante, de una manera similar a la que tiene un coche.

6.1. Control en velocidad de las ruedas de la base móvil

Antes de explicar como se calculan los perfiles de velocidad de las ruedas para hacer que el robot siga una determinada trayectoria usando el algoritmo pure-pursuit conviene recordar algunos de los conceptos expuestos en el apéndice relativo a los perfiles de velocidad disponibles en la PMAC para controlar los motores.

Para que la base siga una determinada trayectoria usando el algoritmo pure-pursuit conviene realizar el control de los motores de las ruedas motrices en velocidad. Un control en velocidad de un motor significa que éste debe girar con el perfil de velocidad angular o lineal que se le indique de forma *indefinida* hasta que se le indique otro. En el control en posición de un motor

lo que se le indica a éste es la distancia que debe recorrer. A diferencia de lo que ocurre en el control en velocidad donde el movimiento del motor no está acotado en el tiempo mientras no se le indique que su velocidad sea 0 cm/s en el control en posición el movimiento sí está acotado en el tiempo. En un control en posición el motor recorrerá la distancia indicada en un intervalo de tiempo mayor o menor dependiendo de otros factores como la velocidad a la que rote o su aceleración, etc, pero antes o después finalizará su recorrido.

La tarjeta controladora PMAC dispone de varios tipos de controles en posición para los motores (comandos `linear`, `pvt`, `rapid`, etc.), sin embargo no posee ningún control en velocidad para los mismos, estrictamente hablando. El control en posición que ofrece la PMAC para los motores no permite hacer un seguimiento de trayectorias adecuado. Esto se debe a que en todos los tipos de controles de posición que la PMAC ofrece cuando se le indica al motor la distancia que debe recorrer ésta no se puede actualizar mientras el motor está en marcha. Esto significa que si se le ha indicado al motor que recorra una distancia X cm y unos instantes de tiempo después se le pide que la distancia a recorrer sea X/2 cm debido a una circunstancia externa, como por ejemplo, que el robot evite un obstáculo, el motor no tendrá en cuenta la distancia X/2 cm, recorrerá la distancia X cm y a continuación, sin pausa, enlazará con el recorrido de la distancia X/2 cm. Claramente este no es el comportamiento deseado cuando se quiere seguir una trayectoria.

Si se quiere obligar a que el motor no recorra la primera distancia indicada, X cm, y recorra la distancia X/2 cm es necesario abortar el movimiento del motor, de modo que éste se para, e indicar la nueva distancia, X/2 cm, tras lo cual el motor inicia el movimiento de nuevo. Claramente este comportamiento tampoco es deseado ya que provoca que el seguimiento de una trayectoria no se aprecie como un movimiento suave sino como un movimiento a "trompicones".

Tras numerosas pruebas con los diferentes tipos de control en posición que ofrece la PMAC se concluyó que ninguno de ellos era válido para realizar un seguimiento de trayectorias dinámico. Afortunadamente la PMAC posee un tipo de control para los motores, que se denomina *Jogging Control*, y que en los manuales de la tarjeta describen como una manera de hacer pruebas con los motores y comprobar que sus parámetros de control[1] son adecuados y garantizan su funcionamiento de forma correcta.

Los comandos de tipo `jog` permiten que los motores se muevan de forma indefinida siguiendo

[1] Cada motor dispone de un centenar de variables que configuran su comportamiento. Estas variables reciben el nombre IXYY, donde la X representa un número de motor y las YY un número de variable.

un perfil de velocidad configurado para cada uno de ellos, es decir, control por velocidad, o bien que recorran una determinada distancia empleando un perfil de velocidad concreto, es decir, control por posición.

Los comandos de tipo `jog` se pueden enviar a la PMAC a través de una interfaz gráfica de usuario presente en la aplicación de control de la tarjeta, denominada PEWIN32Pro (disponible únicamente en S.O Windows), bien a través de un terminal que interpreta comandos, presente en la aplicación anterior o bien a través de comandos ASCII ejecutados dentro de un programa `PLC`. Un comando de tipo `jog` tiene el siguiente aspecto:

```
#<motor_x><tipo_comando_jog>
```

En el comando anterior `<motor_x>` es el número de motor que se quiere mover. El término `<tipo_comando_jog>` se emplea para indicar que tipo de comando `jog` se quiere emplear. Recordemos que los comandos `jog` pueden realizar control en posición o en velocidad de un motor. La secuencia `j<+|->` se usa para hacer control en velocidad del motor y que éste se mueva indefinidamente siguiendo un perfil de velocidad configurado. La parte `<+|->` representa el sentido de giro del motor, `+` para el sentido de giro definido positivo para el motor, `-` para el sentido opuesto. Se pueden concatenar tantos comandos de tipo `jog` en una sola línea como motores a la vez se quieran mover. Por ejemplo, para mover la rueda derecha (motor 1) hacia delante y la rueda izquierda (motor 2) hacia atrás para conseguir una rotación antihoraria indefinida del robot se usa el comando:

```
#1j+#2j-
```

La secuencia `j<+|-><recorrido_en_cm>` realiza un control en posición del motor indicado. El motor se desplazará una distancia `<recorrido_en_cm>`, hacia delante `+`, o hacia atrás `-`, y después se detendrá.

Si recordamos del capítulo 2 un PLC es un programa asíncrono que se ejecuta constantemente sin necesidad de usar un bucle para ello. La PMAC posee la capacidad de ejecutar hasta 32 programas `PLC` a la vez. Se ejecuta el primer programa `PLC`, a continuación el segundo, después el tercero, y así sucesivamente hasta llegar al último programa `PLC` definido. A continuación se inicia la ejecución del primer programa `PLC`, y de este modo procede la tarjeta indefinidamente. Estos programas son muy adecuados para realizar tareas de monitorización de parámetros de la tarjeta de control y para el intercambio periódico de datos entre la PMAC y el PC del robot. Para realizar el intercambio de datos con el PC del robot la PMAC dispone de una memoria de acceso

compartido denominada DPRAM. La DPRAM es una memoria de 16 KB que se haya dividida en segmentos, cada uno de los cuales se usa para intercambiar diferente tipo de información. Ver figura 2.28.

Otro aspecto positivo de los comandos de tipo `jog` es que se pueden actualizar dinámicamente. Por ejemplo si un motor está siguiendo un perfil de velocidad constante en cualquier instante se le puede pedir al motor que modifique dicho perfil para que se ajuste, por ejemplo, a un perfil de velocidad de tipo S, llamado así por el aspecto de S que presenta. Siempre y cuando estos perfiles respeten las configuraciones del motor en cuanto a aceleración máxima, velocidad máxima, etc, ambos perfiles se enlazarán de un modo suave.

El perfil de velocidad que aplica la PMAC a un motor en un comando de tipo `jog` es una función cuadrática que viene determinada por la relación existente entre la velocidad final a la que debe llegar el perfil, V_F, la velocidad inicial en la que empieza el perfil, V_I, el tiempo transcurrido, T, para ir desde V_I hasta V_F y la aceleración Ac usada durante el movimiento. El perfil de velocidad que sigue un motor cuando se usa un comando de tipo `jog` se define con dos parámetros de control. Uno de los parámetros es la velocidad final que debe alcanzar el perfil, V_F. El otro parámetro de control puede ser bien la aceleración máxima en valor absoluto, Ac_{max}, que se debe usar para alcanzar dicha velocidad final o bien el tiempo empleado en alcanzar esa velocidad final, T. Ambos parámetros son equivalentes. La PMAC conoce cual es la velocidad inicial, V_I, con la que empieza el perfil, debido a que es la velocidad final con la que termina el perfil de velocidad anterior. Dependiendo de la relación existente entre los términos V_F, V_I, T y Ac_{max} se consigue que el perfil tenga una apariencia u otra. Para ver los diferentes perfiles de velocidad cuadráticos que se pueden configurar en la PMAC visite el apéndice A.

Uno de los perfiles de velocidad que se puede conseguir a partir de los parámetros V_F, V_I, T y Ac es el *perfil S*. Se trata de un perfil construido con la concatenación de dos funciones cuadráticas. Constituye el perfil más adecuado para controlar un motor en velocidad ya que su derivada es continua y por tanto la aceleración es una función lineal, de este modo el movimiento del motor es suave. Este perfil junto con el perfil de velocidad constante son los únicos que se usan en el algoritmo pure-pursuit de seguimiento de trayectorias desarrollado en este proyecto a partir de comandos de tipo `jog`.

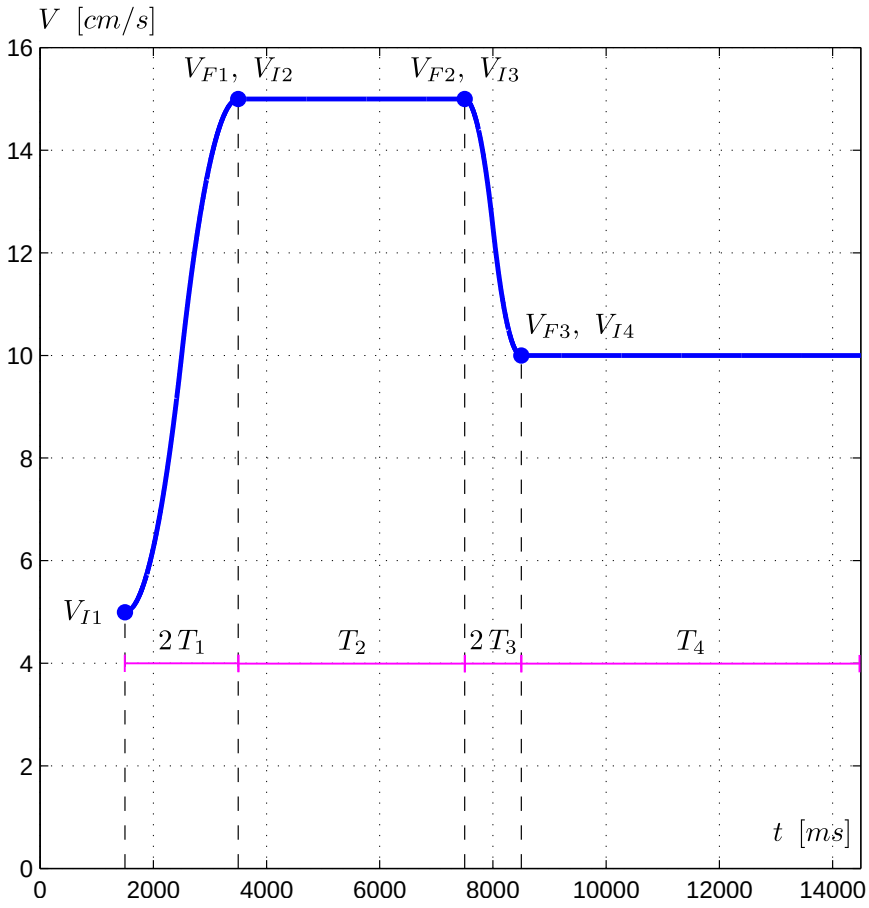

Figura 6.1: Fragmento de perfil de velocidad del motor de una rueda de la base.

En la figura 6.1 se puede apreciar un fragmento de un perfil de velocidad del motor de una rueda de la base que comienza en $V_{I_1} = 5$ cm/s en el instante $t_0 = 1500$ ms. Durante el intervalo de tiempo $T_1 = 1000$ ms la velocidad aumenta de forma cuadrática hasta que en el instante $t_0 + T_1 = 2500$ ms la velocidad vale $\frac{V_{I_1} + V_{F_1}}{2} = 10$ cm/s y la aceleración es máxima, de valor $A_c = \frac{V_{F_1} - V_{I_1}}{T_1} = 10\,\text{cm/s}^2$. Ver figura 6.2. Desde $t_0 + T_1$ ms y durante otros T_1 ms la velocidad sigue aumentando con un perfil cuadrático hasta que en $t_0 + 2\,T_1 = 3500$ ms la velocidad alcanza su valor máximo, $V_{F_1} = 15$ cm/s. Durante este segundo intervalo de tiempo la aceleración ha ido decreciendo desde su valor máximo hasta hacerse nula. A continuación la velocidad se mantiene constante, durante $T_2 = 4000$ ms, al valor alcanzado al final del primer intervalo. Durante el segundo intervalo del movimiento la aceleración es nula. En el tercer intervalo del movimiento la velocidad sigue un perfil cuadrático, similar al usado en el primer intervalo, en este caso descendente. La velocidad disminuye desde $V_{I_3} = 15$ cm/s, en el instante de tiempo

$t_0 + 2\,T_1 + T_2 \;=\; 7500$ ms, hasta $V_{F_3} \;=\; 10$ cm/s, en $t_0 + 2\,T_1 + T_2 + 2\,T_3 \;=\; 8500$ ms. Durante la primera mitad del tercer intervalo la aceleración se va haciendo cada vez más negativa hasta que en $t_0 + 2\,T_1 + T_2 + T_3 \;=\; 8000$ ms ésta tiene un mínimo, de valor $A_c \;=\; \dfrac{V_{F_3} \;-\; V_{I_3}}{T_1} \;=\; -10\,\text{cm/s}^2$. Durante la segunda mitad del tercer intervalo la aceleración aumenta hasta que se hace nula de nuevo en $t_0 \;+\; 2\,T_1 \;+\; T_2 \;+\; 2\,T_3 \;=\; 8500$ ms. Finalmente la velocidad se mantiene constante durante el cuarto intervalo y la aceleración se mantiene nula.

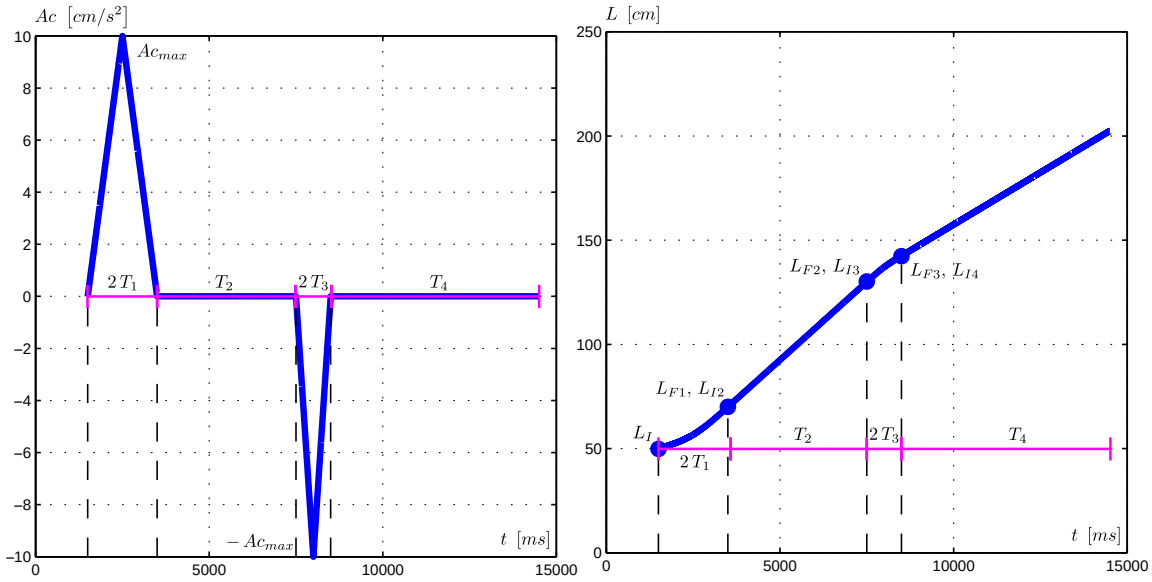

Figura 6.2: Aceleración y recorrido asociado al perfil de velocidad de la figura 6.1.

Sin embargo no todo iban a ser ventajas cuando se usan los comandos de tipo `jog`. Cuando se usan comandos de tipo `jog`, ya sea en la modalidad de control en velocidad o de control en posición, los motores se controlan individualmente. Cuando se usa un control en posición, que no sea del tipo `jog`, por ejemplo comandos `linear`, `pvt`, `rapid`, etc. la PMAC realiza un control coordinado de los motores. Es decir, que todos los motores que deben moverse coordinadamente emplean el mismo tiempo en realizar el movimiento que les corresponde. De este modo aunque se haya configurado un perfil de velocidad diferente para cada motor (no se configura la velocidad final a la que debe llegar un motor, sino el tipo de perfil a usar) y se haya indicado una distancia diferente a recorrer por cada motor, la PMAC ajusta los parámetros necesarios en el perfil de velocidad y en la aceleración máxima de los motores para que recorran la distancia que se les pide en el mismo tiempo. Este comportamiento lo proporciona la PMAC por defecto, sin que sea necesario ninguna tarea por parte del desarrollador, cuando se emplea algún comando de

control en posición de los mencionados arriba. Sin embargo el control individual de los motores que realiza la PMAC en la modalidad de comando `jog` obliga a que sea el desarrollador el que tenga en cuenta todas las variables involucradas en el movimiento de los motores si desea que estos realicen un movimiento coordinado que dure el mismo tiempo. Así en los comandos de tipo `jog` para control en posición se puede hacer que los motores empleen un intervalo de tiempo diferente en completar su recorrido, dependiendo del perfil de velocidad que se les pida, de la aceleración máxima requerida, etc. En los comandos de tipo `jog` para control de velocidad cada motor completa su perfil de velocidad en el tiempo que se le indique y tras completarlo el motor se mantiene a la velocidad alcanzada de manera indefinida hasta que se le indique otro perfil de velocidad.

6.2. Cálculo de parámetros

Una vez que se ha descrito como se van a controlar en velocidad los motores de las ruedas de la base se procede a explicar como se ha implementado el algoritmo pure-pursuit.

Para conseguir que el robot describa una trayectoria arbitraria usando el algoritmo pure-pursuit se dispone de dos programas. Uno de los programas se ejecuta en el PC del robot, el otro es un PLC ejecutado en la tarjeta PMAC. El programa ejecutado en el PC envía información periódicamente al PLC, cada $T_{MOV} = 250\,\text{ms}$, a través de la memoria compartida DPRAM.

En primer lugar se van a describir las tareas que realiza el programa ejecutado en el PC del robot.

Paso 1: Obtener la ubicación del marco de referencia `link_base` en el entorno de trabajo del algoritmo de localización global evolutivo diferencial que se ejecuta en el robot ([1]). Cada vez que el algoritmo pure-pursuit necesita conocer la ubicación del robot solicita al algoritmo de localización que le proporcione dicha información. Esta información es intercambiada entre procesos a través de servicios del framework ROS. Una vez que el algoritmo de localización global ha encontrado la ubicación del robot nuevas relocalizaciones tardan el orden de unas pocas decenas de milisegundos.

Paso 2: Determinar el segmento perteneciente a la trayectoria de interés donde se encuentra la proyección ortogonal de la posición del robot. Este segmento, conocido como segmento relevante, viene determinado por los dos puntos de control de la trayectoria de interés que lo delimitan, i e $i+1$. Por tanto para determinar el segmento relevante habrá que hallar los puntos de control

i e $i+1$ entre los que se encuentra la proyección perpendicular de la posición del marco de referencia `link_base`. Para obtener los puntos de control i e $i+1$ se usa el algoritmo de la figura 3.6.

Paso 3: A partir de los puntos de control seleccionados en el paso anterior encontrar el primer punto de control en la trayectoria de interés que se encuentra a una distancia L_{ah} del marco de referencia `link_base`. Una vez seleccionado el punto de control destino se halla el ángulo θ_{err} con el que se puede calcular del radio de giro, R_{sF}, que permite trazar un arco que une la posición actual del robot con el punto de control destino.

A continuación se va a detallar que valor debe tener el parámetro L_{ah}. Para satisfacer el requisito de que el robot se desplace siempre hacia delante, como un coche, se debe cumplir que $\forall t_k \geq 0$:

$$v_d\,(t_k) \geq 0 \tag{6.2}$$

$$v_i\,(t_k) \geq 0 \tag{6.3}$$

La PMAC se va a encargar de hacer el control de los perfiles S de velocidad de los motores de las ruedas a partir de comandos `jog`. Por tanto el programa ejecutado en el PC debe comunicarle periódicamente a la PMAC, a través de la DPRAM, las velocidades finales de cada uno de los motores de la base, V_{dF} y V_{iF}, en cm/s, así como el tiempo que debe emplear cada motor en completar la mitad de su perfil, T_d y T_i, en ms. Cada $T_{MOV} = 250\,\text{ms}$ el PC deposita nueva información en la DPRAM. Esto significa que el perfil S de velocidad de cada rueda debe completarse como máximo en el intervalo T_{MOV}, es decir:

$$2 \cdot T_d \leq T_{MOV} \tag{6.4}$$

$$2 \cdot T_i \leq T_{MOV} \tag{6.5}$$

Por tanto, si por ejemplo, el perfil S de velocidad del motor de una rueda se completa en $2 \cdot T < T_{MOV}$, en el intervalo de tiempo restante, $T_{MOV} - 2 \cdot T$, el motor mantiene la velocidad alcanzada al final del perfil S hasta que se completa el intervalo T_{MOV}.

A partir de las expresiones 4.4 y 4.5, vistas en el capítulo dedicado a la odometría (capítulo 4), las expresiones de las velocidades finales de los motores de las ruedas, V_{dF} y V_{iF}, se pueden

escribir como sigue:

$$V_{dF} = \left(1 + \frac{r_b}{R_{sF}}\right) \cdot V_{bF}\left(|R_{sF}|\right) \tag{6.6}$$

$$V_{iF} = \left(1 - \frac{r_b}{R_{sF}}\right) \cdot V_{bF}\left(|R_{sF}|\right) \tag{6.7}$$

Para que las expresiones 6.6 y 6.7 sean positivas, y así satisfacer el requisito de que el robot se desplace siempre hacia delante, se asegura que los dos factores en cada una de ellas sean positivos. En primer lugar se va a estudiar el comportamiento de los términos $\left(1 \pm \frac{r_b}{R_{sF}}\right)$. Como se puede apreciar en la figura 6.3 ambos términos se comportan de un modo similar:

$$\left(1 + \frac{r_b}{R_{sF}}\right) \geq 0 \quad \forall R_{sF} \in (-\infty, -r_b] \cup [0, \infty)$$

$$\left(1 - \frac{r_b}{R_{sF}}\right) \geq 0 \quad \forall R_{sF} \in (-\infty, 0] \cup [r_b, \infty)$$

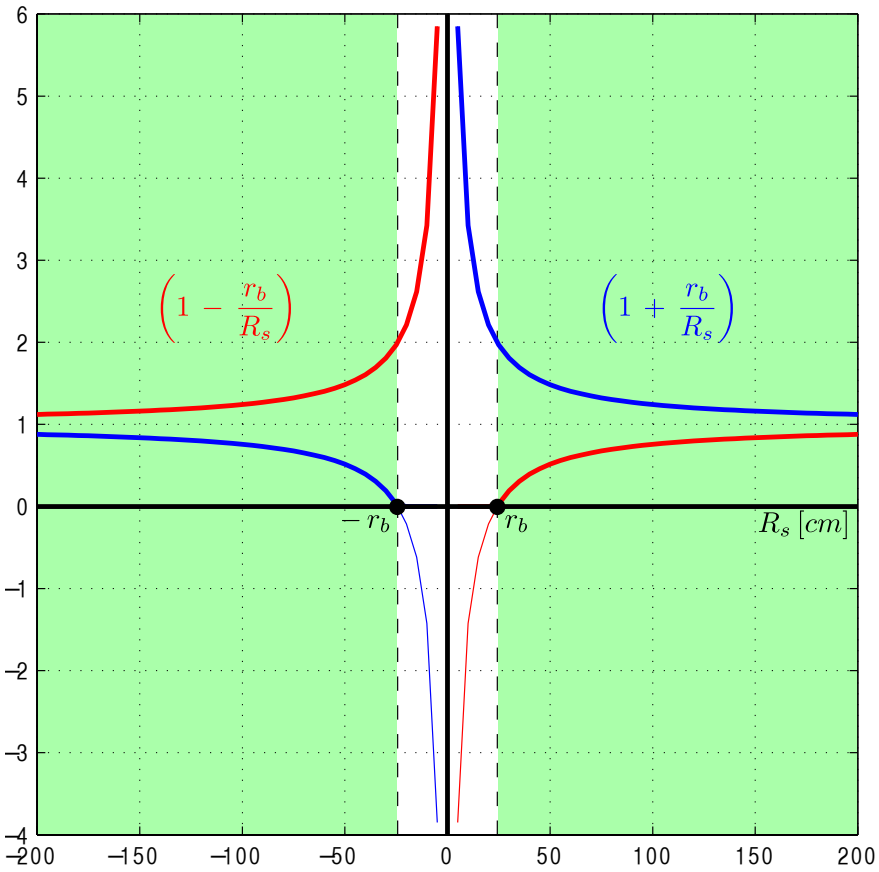

Figura 6.3: Comportamiento de los términos $\left(1 \pm \frac{r_b}{R_{sF}}\right)$ en función de R_{sF}

Observando la figura 6.3 se puede afirmar que los términos $\left(1 \pm \frac{r_b}{R_{sF}}\right)$ son ambos positivos cuando:

$$R_{sF} \in (-\infty, -r_b] \ \cup \ [r_b, \infty) \tag{6.8}$$

es decir

$$|R_{sF}| \geq r_b \tag{6.9}$$

A continuación se averiguará cual es el rango de valores que debe tener la distancia L_{ah} para satisfacer la condición 6.9.

$$|R_{sF}| = \left|\frac{L_{ah}}{2 \cdot \sin(\theta_{err})}\right| = \frac{L_{ah}}{2 \cdot |\sin(\theta_{err})|} \geq r_b \tag{6.10}$$

Se ha tenido en cuenta que la distancia L_{ah} es siempre positiva, de ahí que $|L_{ah}| = L_{ah}$. Despejando el término L_{ah} de la expresión anterior se obtiene:

$$L_{ah} \geq 2 \cdot r_b \cdot |\sin(\theta_{err})| \tag{6.11}$$

$$2 \cdot r_b \cdot |\sin(\theta_{err})| \leq 2 \cdot r_b \tag{6.12}$$

En la expresión 6.12 se ha tenido en cuenta que $|\sin(\theta_{err})| \leq 1$. Por tanto siempre que se deba elegir un nuevo punto de control destino en la trayectoria de interés se seleccionará el primero para el que se verifique que:

$$L_{ah} \geq 2 \cdot r_b \tag{6.13}$$

A continuación se describen cuales son las expresiones que definen la función a trozos $V_{bF}(|R_{sF}|)$. Mientras que los términos $\left(1 \pm \frac{r_b}{R_{sF}}\right)$ determinan la relación que debe existir entre las velocidades las ruedas para describir una trayectoria circular de radio R_{sF}, el término $V_{bF}(|R_{sF}|)$ determina cuan rápido debe circular el sistema robótico por la trayectoria, es decir:

Cuanto mayor sea el radio de giro, y por lo tanto más parecida sea la trayectoria a una línea recta, mayor es la velocidad a la que puede desplazarse el robot. En cambio, cuanto menor es el radio de giro, y por tanto más cerrada es la circunferencia que el robot debe realizar, menor es la velocidad a la que éste debe desplazarse.

Con estas premisas la expresión a trozos queda:

$$V_{bF}(|R_{sF}|) = \begin{cases} \left(\frac{V_2 - V_1}{9 \cdot r_b}\right)(|R_{sF}| - r_b) + V_1 & si \quad r_b \leq |R_{sF}| \leq 10 \cdot r_b \\ V_2 & si \quad |R_{sF}| \geq 10 \cdot r_b \end{cases} \tag{6.14}$$

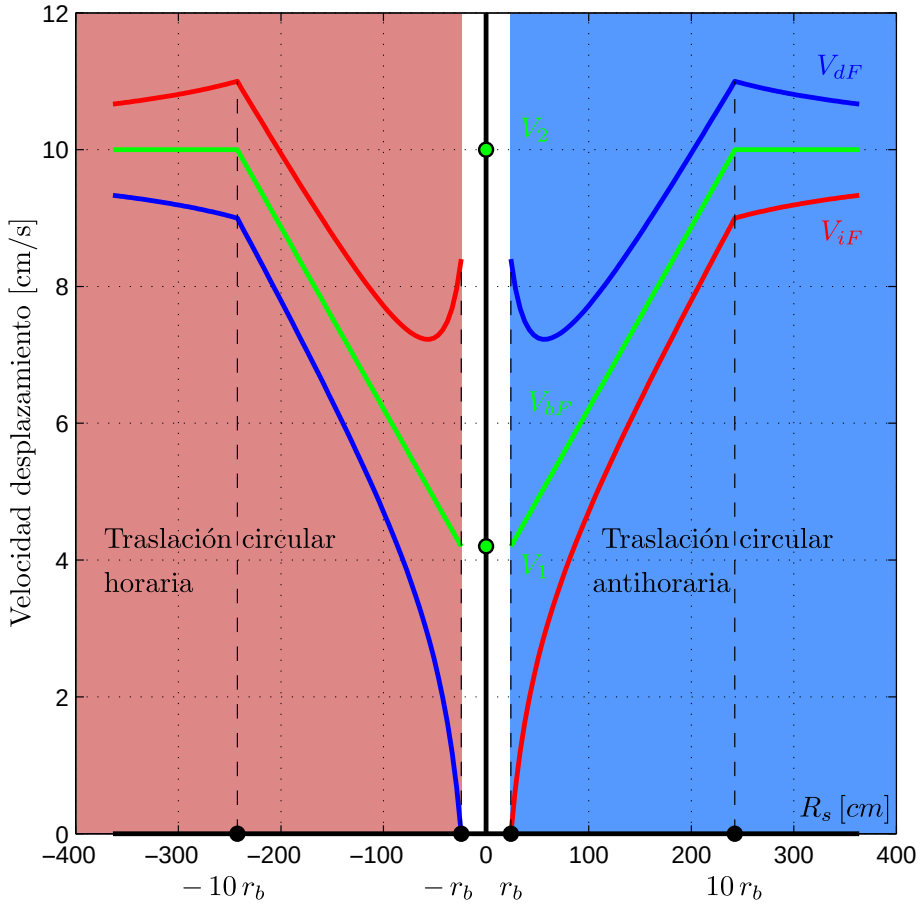

Figura 6.4: V_{dF} y V_{iF} en función de R_{sF}.

La velocidad máxima de desplazamiento lineal que puede aplicarse a una rueda es 25 cm/s, sin embargo, experimentalmente se ha comprobado que cuando el robot se desplaza en línea recta, un compromiso adecuado entre estabilidad de la plataforma y rapidez de movimiento se alcanza cuando se usa una velocidad de desplazamiento lineal de $V_{bF} = 10$ cm/s. Cuanto mayor es el radio de giro más próxima está la velocidad final de desplazamiento lineal de la rueda derecha e izquierda del valor V_2, de manera que cuando la base se desplaza en línea recta, y por tanto el radio de giro es infinito, se cumple que $V_{dF} = V_{iF} = V_2$. Por tanto se escoge el valor $V_2 = 10$ cm/s. Cuando el robot usa el radio de giro mínimo, r_b, una de las ruedas permanece parada y la otra gira, haciendo que el robot describa una circunferencia cuyo centro es la rueda parada.

En estas circunstancias se ha verificado experimentalmente que un compromiso aceptable entre la estabilidad de la plataforma robótica y la rapidez de movimiento se alcanza al usar la velocidad

angular $\omega_b = 10°/\text{s}$.

$$V_1 = V_{bF} = |\omega_b \cdot R_{sF}| = |\omega_b| \cdot r_b = \frac{10 \cdot \pi}{180} \cdot r_b = 4'356\,\text{cm/s} \tag{6.15}$$

Recapitulando, hasta el momento en el paso 3 se ha averiguado que el siguiente punto de control destino para el robot debe estar separado de su posición actual por una distancia de $L_{ah} = d_b\,\text{cm}$ (distancia entre ruedas de la base). Del paso 2 se conocen los puntos de control i e $i+1$ de la trayectoria de interés entre los que se encuentra la proyección perpendicular de la ubicación del marco de referencia `link_base`. Se calcula la distancia euclídea entre la posición del marco de referencia anterior y el extremo $i+1$. Esta distancia es el término L_{ah}. Si se cumple que $L_{ah} \geq 2 \cdot r_b$ entonces se ha encontrado el punto destino. En cambio si ocurre que $L_{ah} < 2 \cdot r_b$, entonces se calcula la distancia euclídea entre la posición del marco `link_base` y el punto $i+2$, que pertenece al segmento $i+1$. Si en esta ocasión se cumple que $L_{ah} \geq 2 \cdot r_b$ entonces se ha encontrado el punto destino. En cambio si ocurre que $L_{ah} < 2 \cdot r_b$ entonces se calcula una nueva distancia euclídea usando el punto $i+3$, perteneciente al segmento $i+2$. Este procedimiento se repetiría hasta encontrar el primer punto de la trayectoria para el cual la distancia euclídea calculada sea mayor o igual que el diámetro de la base del robot. A continuación se calcula el radio de giro, R_{sF}, que permite al robot describir un arco que une su ubicación actual con el punto de control destino. Debido a la condición anterior, $L_{ah} \geq 2 \cdot r_b$, se satisface que $|R_{sF}| \geq r_b$. A partir del radio de giro se calcula la velocidad final de desplazamiento lineal que debe alcanzar cada rueda, V_{dF} y V_{iF}, partiendo de la velocidad inicial que lleven, V_{dI} y V_{iI}, siguiendo un perfil S que asegure que su aceleración sea continua. Debido a que el radio de giro en valor absoluto es mayor o igual que el radio de la base la velocidad final de las ruedas se mantiene positiva permanentemente, lo que asegura que el robot se desplaza siempre hacia delante, cumpliendo el requisito enunciado al principio del capítulo. El perfil S de velocidad de cada rueda debe completarse en un período de tiempo inferior o igual a T_{MOV}. Si un perfil S de velocidad se completa en un periodo de tiempo inferior a T_{MOV} en el periodo de tiempo restante hasta alcanzar dicho valor la rueda gira con velocidad constante e igual a la adquirida al final del perfil S completado. Tras el intervalo de tiempo T_{MOV} el algoritmo comienza de nuevo, solicitando al algoritmo de localización la ubicación del robot. A este paso le siguen los pasos 2 y 3 explicados anteriormente.

El PC debe comunicarle a la PMAC, a través de la DPRAM, las velocidades finales de ambas ruedas, así como el tiempo que debe emplearse en completar la mitad de cada perfil S.

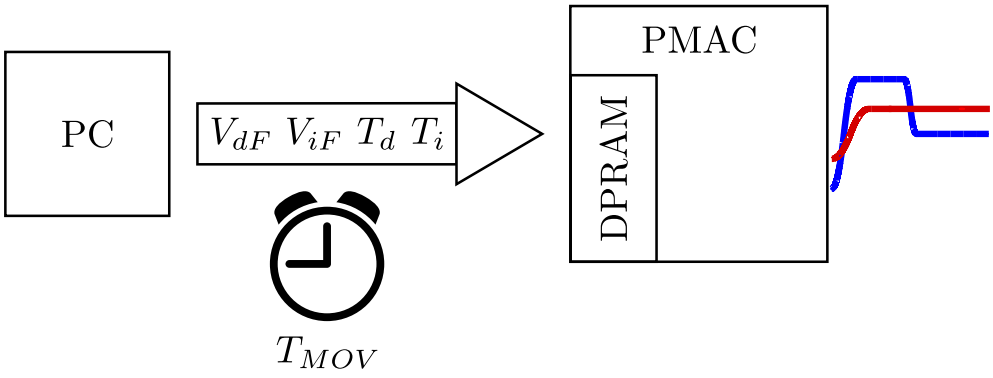

Figura 6.5: Comunicación de parámetros del PC a la PMAC cada T_{MOV} ms.

Del apéndice A se sabe que la aceleración máxima, en valor absoluto, que experimenta un motor cuando la PMAC le aplica un perfil S de velocidad viene dada por la expresión:

$$Acel_{max} = \frac{|V_F - V_I|}{T} = \frac{|\Delta V|}{T} \tag{6.16}$$

Experimentalmente se ha comprobado que cuando se usa una aceleración máxima en los motores de las ruedas de $Acel_{MAX} = 10\,\mathrm{cm/s^2}$ el robot no da trompicones al iniciar la marcha, iniciándose ésta de manera suave. Este valor se considera seguro y suficiente para las necesidades del algoritmo, al menos por el momento, así pues, desde el algoritmo ejecutado en el PC se debe asegurar que la aceleración que se aplica al motor de cada rueda no excede nunca $Acel_{MAX}$.

$$Acel_{max} \leq Acel_{MAX} \tag{6.17}$$

Para garantizar que la aceleración que la PMAC aplica al motor de cada rueda es inferior o igual al umbral máximo indicado se calcula el valor de T, para cada rueda, del siguiente modo:

$$T = ceil\left(\frac{|\Delta V|}{Acel_{MAX}}\right) \tag{6.18}$$

donde la función `ceil` proporciona el valor entero más cercano que sea mayor o igual a su argumento. La función `ceil` es necesaria por dos motivos, en primer lugar, para no exceder la aceleración máxima aplicable al motor de cada rueda, y en segundo lugar porque la PMAC necesita que el valor de T sea un número entero, expresado en ms. El término V_F se calcula con la expresión 6.6 o 6.7, dependiendo de la rueda para el que se esté hallando, y el término V_I es la velocidad alcanzada al final del perfil anterior para dicha rueda. De este modo, necesariamente:

$$Acel_{max} = \frac{|\Delta V|}{ceil\left(\frac{|\Delta V|}{Acel_{MAX}}\right)} \leq Acel_{MAX} \tag{6.19}$$

Dado que el denominador en la expresión 6.18 es constante, el valor de T es proporcional al valor de $|\Delta V|$. Ver tramo 3 de la figura 6.6. Así pues la aceleración máxima real, en valor absoluto, que experimenta el motor de una rueda cuando se le aplica un perfil S de velocidad presenta el aspecto del tramo 3 de la figura 6.7. El efecto diente de sierra de la figura 6.7 es debido a la efecto escalón introducido por la función `ceil` en el cálculo de T.

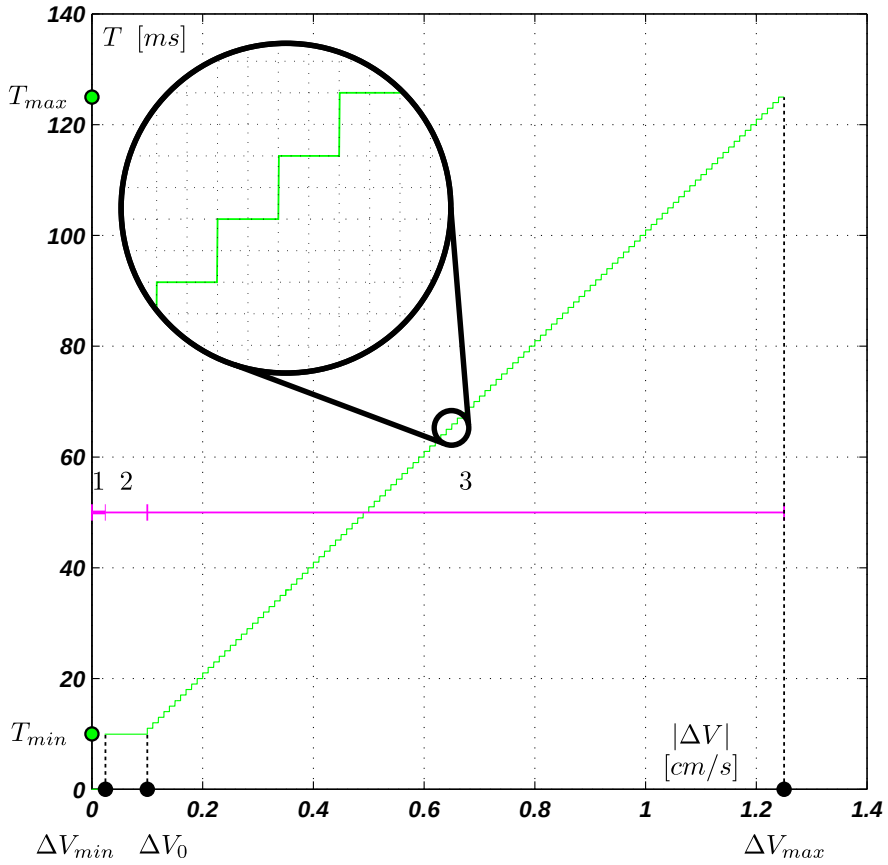

Figura 6.6: Tiempo que se tarda en realizar la mitad de un perfil S de velocidad.

El periodo de tiempo máximo que puede durar un perfil S de velocidad está acotado por el intervalo de actualización del algoritmo pure-pursuit desarrollado, es decir, T_{MOV}. Así pues:

$$T_{max} = \frac{T_{MOV}}{2} = \frac{250}{2} = 125 \text{ ms} \tag{6.20}$$

Por tanto, aplicando un perfil S de velocidad a una rueda durante un período de tiempo T_{MOV} y usando una aceleración máxima, en valor absoluto, de valor $Acel_{MAX}$ la variación máxima de

velocidad que se consigue es:

$$\Delta V_{max} \;=\; T_{max} \cdot Accel_{MAX} \;=\; 0'125 \cdot 10 \;=\; 1'25\,\mathrm{cm/s} \tag{6.21}$$

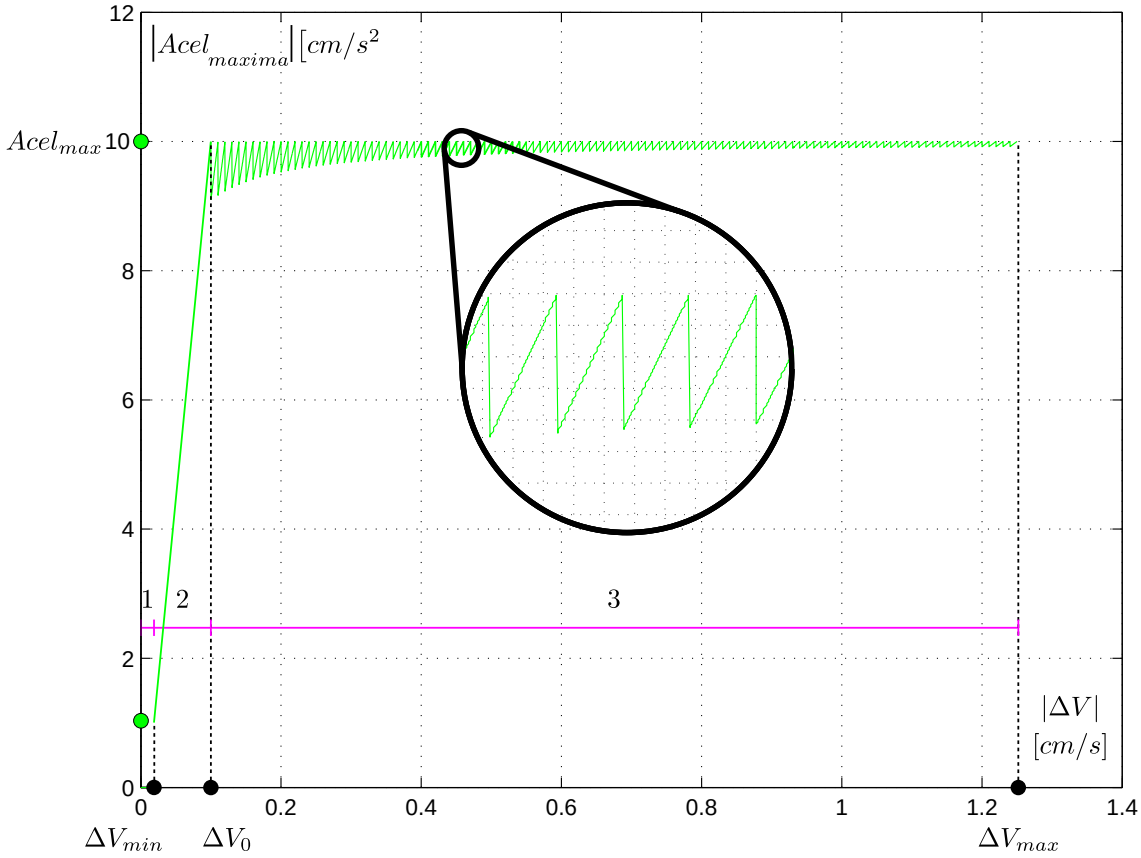

Figura 6.7: Aceleración máxima real, en valor absoluto, aplicada al motor de una rueda cuando sigue un perfil S de velocidad.

De este modo cualquier valor de V_F hallado para una rueda, bien con la expresión 6.6 o 6.7, para el que se cumple que $|\Delta V| > \Delta V_{max}$ se debe rectificar a:

$$V_F \;=\; V_I \;+\; sign(\Delta V) \cdot \Delta V_{max} \tag{6.22}$$

ya que éste es el valor de V_F que puede alcanzarse en el intervalo de tiempo T_{MOV} a partir de una velocidad inicial V_I dada. Así pues si $|\Delta V| > \Delta V_{max}$, tras la corrección del valor de V_F se obtiene $|\Delta V| = \Delta V_{max}$. Si se rectifica la velocidad final de una o ambas ruedas la relación resultante entre éstas, en general, no coincide con la relación teórica necesaria para obtener el radio de giro R_{sF} que permite al robot ir desde su posición actual a la posición destino. Por

tanto en la siguiente iteración del algoritmo se hará la corrección necesaria en el arco a trazar por el robot para llegar al punto destino seleccionado en esa iteración. Estas restricciones surgen debido a las limitaciones físicas del robot o a los umbrales escogidos para algunos parámetros del algoritmo.

Tal y como indica la expresión 6.18, dado que su denominador permanece constante, a medida que disminuye el valor del término $|\Delta V|$ también disminuye el valor del parámetro T.

El período de tiempo mínimo que puede durar un perfil S de velocidad es 2 ms, es decir, 1 ms por cada una de las dos partes que conforman el perfil.

$$2 \cdot T_{MIN} = 2\,\text{ms} \tag{6.23}$$

$$T_{MIN} = 1\,\text{ms} \tag{6.24}$$

En estas condiciones un perfil S de velocidad pierde su forma característica de "S" y se convierte en un perfil lineal. En la práctica no es aconsejable realizar perfiles de velocidad de esa duración, pues se está forzando a la PMAC a su máxima capacidad de funcionamiento, lo cual puede proporcionar resultados o fallos imprevistos. Por tal motivo se ha decidido usar un período de tiempo mínimo para completar un perfil S de velocidad que tenga en cuenta una tolerancia para el funcionamiento de la PMAC.

$$2 \cdot T_{min} = 10 \cdot (2 \cdot T_{MIN}) = 20\,\text{ms} \tag{6.25}$$

$$T_{min} = 10\,\text{ms} \tag{6.26}$$

Por tanto la variación mínima de velocidad que se puede conseguir usando T_{min} y $Acel_{MAX}$ es:

$$\Delta V_0 = T_{min} \cdot Acel_{MAX} = 0'1\,\text{cm/s} \tag{6.27}$$

Para variaciones de velocidad, en valor absoluto, que se encuentren por debajo de ΔV_0 no se puede usar el término $Acel_{MAX}$ para hallar el valor de T, ya que este valor se hallaría por debajo de T_{min}. En estas circunstancias se deja fijo del valor de T a T_{min}. Así, a medida que disminuye la variación de velocidad requerida disminuye la aceleración máxima, en valor absoluto, aplicada al motor. Tramo 2 de la figura 6.7. Se ha considerado una aceleración mínima de $Acel_{MIN} = 1\,\text{cm/s}^2$. De este modo la variación mínima de velocidad que se puede considerar es:

$$\Delta V_{min} = T_{min} \cdot Acel_{MIN} = 0'01\,\text{cm/s} \tag{6.28}$$

lo cual resulta más que suficiente para las necesidades del algoritmo desarrollado.

Así pues valores del término $|\Delta V|$ inferiores a ΔV_{min} son truncados a $|\Delta V| = 0\,\text{cm/s}$ al considerar que dichos valores originales no introducen apenas modificaciones en el camino descrito por el robot. En estas circunstancias no hay perfil S que aplicar a la rueda usada, se mantiene, durante un período de tiempo T_{MOV}, la velocidad alcanzada al final del perfil de velocidad anterior . Ver tramo 1 de las figuras 6.6 y 6.7.

Resumiendo:

1. Si $|\Delta V| >= 0\,\text{cm/s}$ y $|\Delta V| <= \Delta V_{min} \longrightarrow |\Delta V| = 0\,\text{cm/s}$, $T = 0\,\text{ms}$. No hay perfil S de velocidad. Sólo existe un perfil de velocidad constante, cuyo valor es la velocidad alcanzada al final del perfil de velocidad anterior.

2. Si $|\Delta V| >= \Delta V_{min}$ y $|\Delta V| <= \Delta V_0 \longrightarrow T = T_{min}$. Perfil S de velocidad de duración fija.

3. Si $|\Delta V| >= \Delta V_0$ y $|\Delta V| <= \Delta V_{max} \longrightarrow T = ceil\left(\frac{|\Delta V|}{Acel_{MAX}}\right)$. Perfil S de velocidad de duración variable.

A continuación se explicará un ejemplo con el objetivo de ilustrar todos los conceptos expuestos anteriormente. En la figura 6.8 se pueden apreciar numerosos radios de giro que deben conseguirse en instantes de tiempo concretos para trazar una determinada trayectoria. A modo de ejemplo se ilustra el radio de giro calculado en el instante $t = 2500\,\text{ms}$, $R_{sF} = 32'98\,\text{cm}$.

En la figura 6.9 se puede observar la velocidad final de desplazamiento lineal de la rueda derecha e izquierda (puntos de diverso color) que deben conseguirse en instantes de tiempo concretos para obtener los radios de giro mostrados en la figura 6.8. Obviamente para que el motor de cada rueda pueda pasar de la velocidad inicial que lleva, V_I, a la velocidad final calculada, V_F, se debe emplear un perfil S de velocidad que dura una determinada cantidad de tiempo. Este tipo de perfil asegura que la aceleración aplicada al motor es lineal y continua. El motor de cada rueda dispone de T_{MOV} ms para alcanzar su velocidad final. Si el motor de una rueda alcanza su velocidad final antes de que finalice dicho intervalo mantiene esa velocidad final hasta que finalice el mismo.

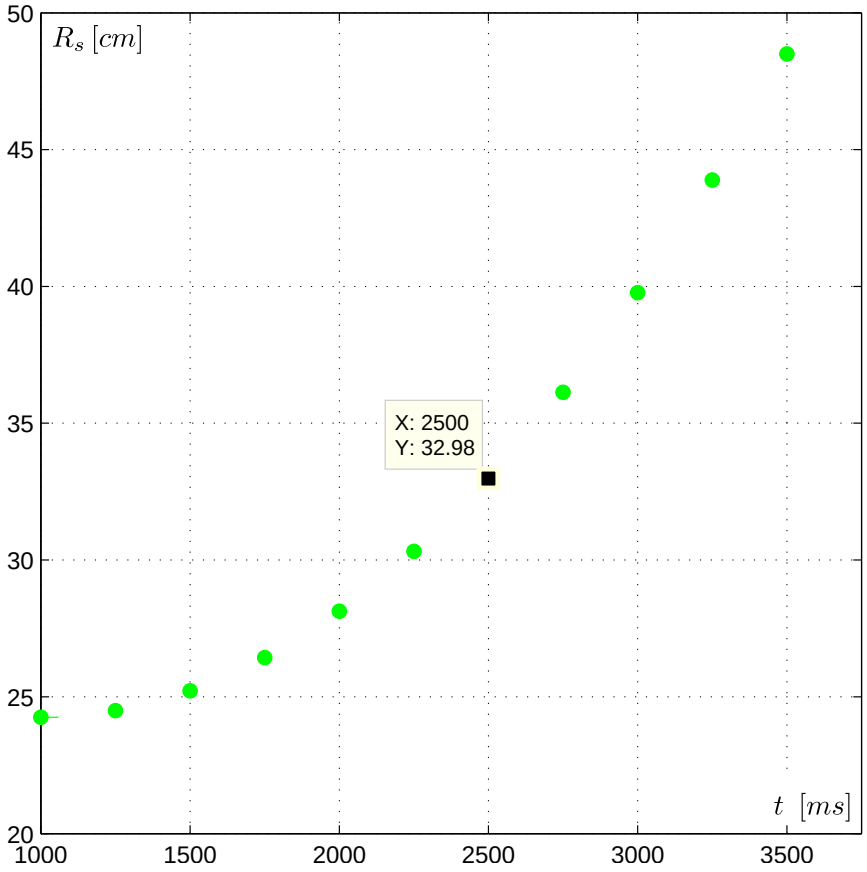

Figura 6.8: Radios de giro calculados cada $T_{MOV} = 250$ ms, necesarios para trazar una trayectoria concreta.

Por ejemplo, en la figura 6.9 se puede apreciar como a la función $v_d\,(t)$ le lleva 34 ms pasar de la velocidad inicial que lleva en $t = 2500$ ms, $V_{dI} = 7'907$ cm/s, a la velocidad final calculada en ese instante, $V_{dF} = 7'745$ cm/s. En los restantes 216 ms, hasta completar el período de tiempo T_{MOV}, se mantiene la velocidad final V_{dF} alcanzada por el motor de la rueda derecha. En cambio a la función $v_i\,(t)$ le lleva 62 ms pasar de la velocidad inicial que lleva en $t = 2500$ ms, $V_{iI} = 0'878$ cm/s a la velocidad final calculada para ese mismo instante, $V_{iF} = 1'181$ cm/s. En los restantes 188 ms se mantiene la velocidad V_{iF} alcanzada por el motor de la rueda izquierda.

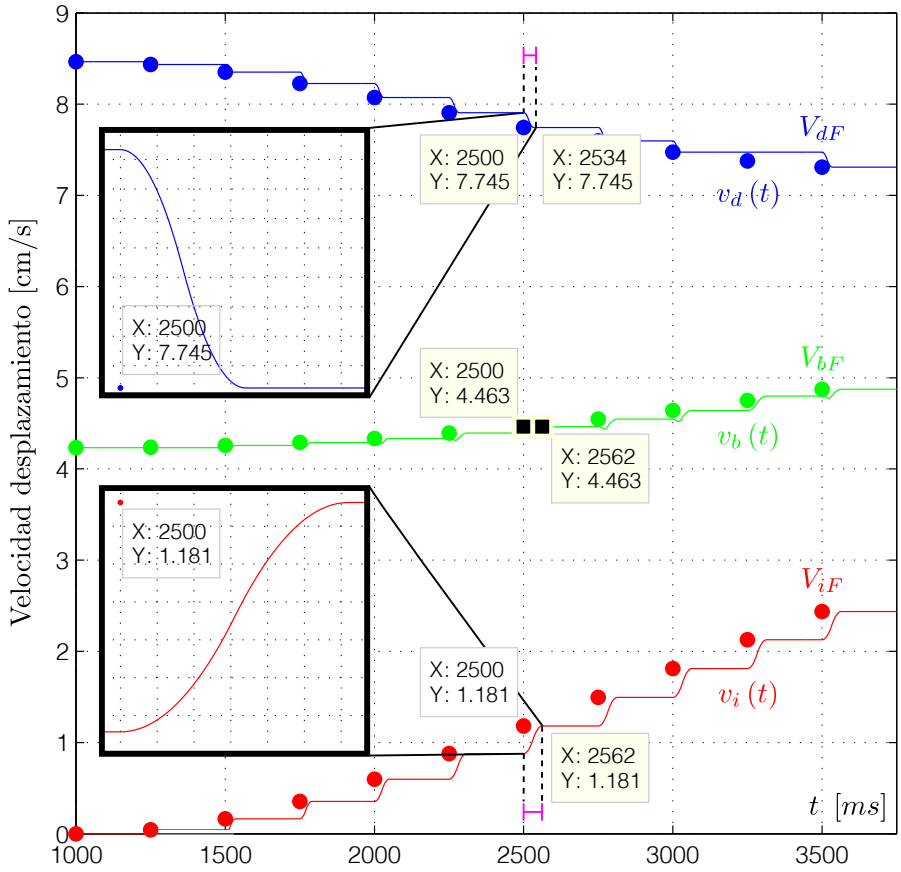

Figura 6.9: Evolución de la velocidad del motor de la rueda derecha e izquierda, $v_d\,(t)$ y $v_i\,(t)$, para obtener los radios de giro de la figura 6.8.

Obviamente el radio de giro deseado sólo se logra cuando los motores de ambas ruedas alcanzan la velocidad final calculada para ellos. Según ambos motores alcanzan su velocidad final correspondiente el radio de giro se aproxima a su valor final, tal y como puede apreciarse en la figura 6.10. El radio de giro final calculado para el instante $t = 2500$ ms es $R_{sF} = 32'98$ cm, mientras que la función radio de giro, en dicho instante es, $R_s\,(t = 2500\,\text{ms}) = 30'31$ cm. Dado que a la función $v_d\,(t)$ le lleva 34 ms alcanzar su velocidad final, V_{dF}, y a la función $v_i\,(t)$ le lleva 62 ms alcanzar su velocidad final, V_{iF}, a la función $R_s\,(t)$ le lleva 62 ms alcanzar el valor R_{sF}. En los restantes 188 ms, hasta completar el intervalo de tiempo T_{MOV} ms, se mantiene el radio de giro R_{sF}.

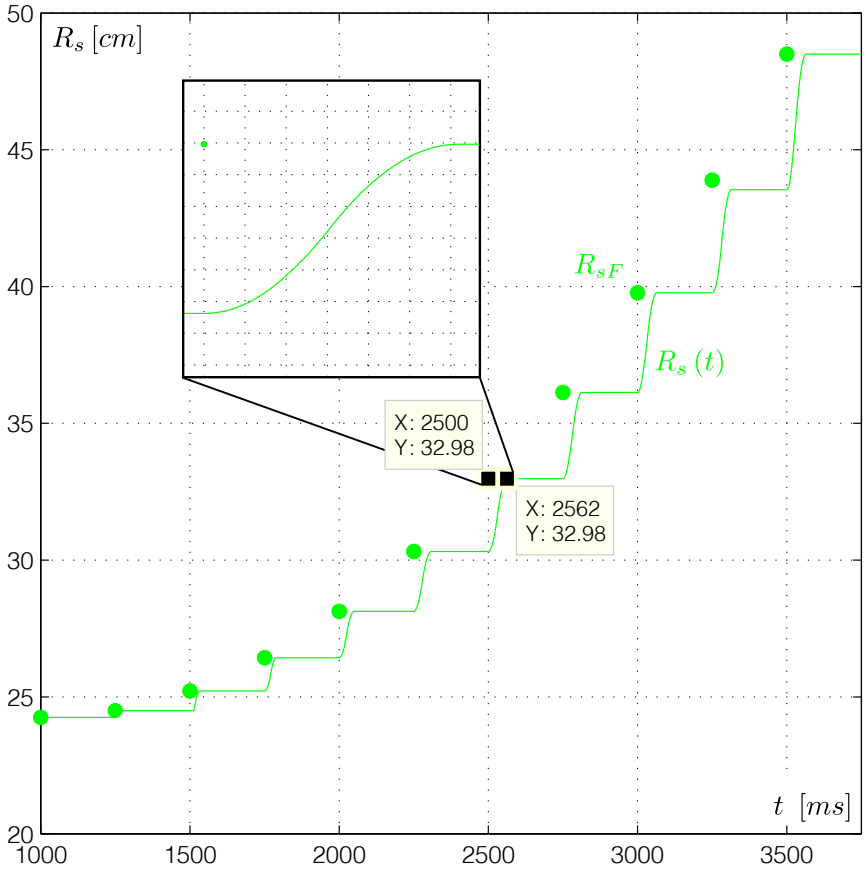

Figura 6.10: Evolución del radio de giro $R_s\,(t)$.

Si alguno de los motores de las ruedas no puede alcanzar la velocidad final que se le pide en el intervalo de tiempo T_{MOV}, debido a que $|\Delta V| > \Delta V_{max}$, entonces se trunca su valor a $V_F = V_I + sign(\Delta V) \cdot \Delta V_{max}$. En estas circunstancias, el radio de giro necesario, R_{sF}, no se logrará perfectamente en el intervalo de tiempo T_{MOV}. En el próximo intervalo temporal se hará la corrección oportuna de las velocidades de los motores de las ruedas para compensar esta desviación introducida.

Por último cabe mencionar que valores de $|\Delta V|$ inferiores a ΔV_{min} son truncados a $|\Delta V| = 0$, es decir, $V_F = V_I$.

A continuación se describe el proceso de parada que efectúa el algoritmo para detener al robot en el último punto de la trayectoria de interés. Cuando se detecta el último punto de la trayectoria de interés el marco de referencia `link_base` se encuentra a una distancia inferior a d_b de ese punto, $L_{ah} < d_b$. Con la distancia al punto objetivo se calculan los términos θ_{err} y R_{sF}. Anteriormente se ha indicado que para que el algoritmo haga que el robot se desplace siempre hacia delante se

debe asegurar que el radio de traslación circular del marco de referencia `link_base` alrededor del C.R.I es mayor o igual a r_b. Esta condición se cumple siempre asegurando que la distancia que separa el marco de referencia `link_base` del punto de control seleccionado en la trayectoria de interés es mayor o igual a d_b. Esta última condición es suficiente pero no necesaria, es decir, puede suceder que aunque la distancia que separa al marco de referencia `link_base` del punto de control seleccionado en la trayectoria de interés sea inferior a d_b el término R_{sF} sea ser mayor o igual a r_b. Si en esta ocasión el término R_{sF} resultante es inferior a r_b su valor se trunca a r_b, lo que garantiza la forma de desplazamiento deseada del robot.

Con el término R_{sF} se puede calcular cual es el arco que debe recorrer cada rueda para que la marcha del robot finalice en el último punto de la trayectoria de interés.

$$arco_d = (R_{sF} + r_b) \cdot 2 \cdot \theta_{err} \tag{6.29}$$

$$arco_i = (R_{sF} - r_b) \cdot 2 \cdot \theta_{err} \tag{6.30}$$

El recorrido que cada rueda realiza durante un peril S de velocidad es $\Delta L = (V_F + V_I) \cdot T$. Se conoce la velocidad inicial de desplazamiento de cada rueda, la velocidad final de desplazamiento de cada rueda debe ser nula y también se conoce la distancia que debe recorrer cada una de ellas. Por tanto el valor de T es:

$$T = ceil\left(\frac{(R_{sF} \pm r_b) \cdot 2 \cdot \theta_{err}}{V_I}\right) \tag{6.31}$$

donde el signo + se usa para la rueda derecha y el signo - para la rueda izquierda. De nuevo se usa la función ceil para granizar que el valor del término T es un número entero. Se recuerda que el valor de T debe estar expresado en milisegundos.

A continuación se comprueba si la aceleración máxima, en valor absoluto, que experimentará cada motor de la base durante el perfil S de velocidad que se le aplicará excede el valor $Acel_{MAX}$. Si para alguno de los motores de las ruedas la aceleración máxima, en valor absoluto, que experimentará es mayor que $Acel_{MAX}$ se recalcula el intervalo de tiempo T que debe durar la mitad del perfil S de velocidad que se le aplicará para no exceder este umbral máximo. Obviamente este nuevo intervalo de tiempo, necesariamente mayor que el calculado en primera instancia, provoca que el arco que recorre la rueda es mayor del necesario, aunque la desviación es insignificante, en el caso de producirse.

$$\text{Si } T = ceil\left(\frac{(R_{sF} \pm r_b) \cdot 2 \cdot \theta_{err}}{V_I}\right) < ceil\left(\frac{|-V_i|}{Acel_{MAX}}\right) \longrightarrow T = ceil\left(\frac{|-V_i|}{Acel_{MAX}}\right) \tag{6.32}$$

A continuación se le transfiere a la PMAC, a través de la DPRAM, los términos $V_{dF} = 0\,\text{cm/s}$, $V_{iF} = 0\,\text{cm/s}$, T_d y T_i. Cuando el perfil S de velocidad de cada motor de la base móvil se completa el robot se detiene.

Para finalizar el capítulo se explica como se orienta robot inicialmente hacia la trayectoria antes de iniciar su recorrido. El algoritmo de planificación de trayectorias utilizado construye una ruta óptima, en base a algún criterio establecido, entre la posición actual del robot y un punto destino de interés. El algoritmo que construye la trayectoria tiene en cuenta la posición del robot, pero no su orientación, lo que puede hacer que ésta se encuentre situada a la espalda del robot como sucede en la figura 6.11. En estas circunstancias lo más razonable es que el robot se oriente hacia la trayectoria, mediante un movimiento de rotación sobre sí mismo, y continuación incie el recorrido de la trayectoria usando el algoritmo pure-pursuit implementado.

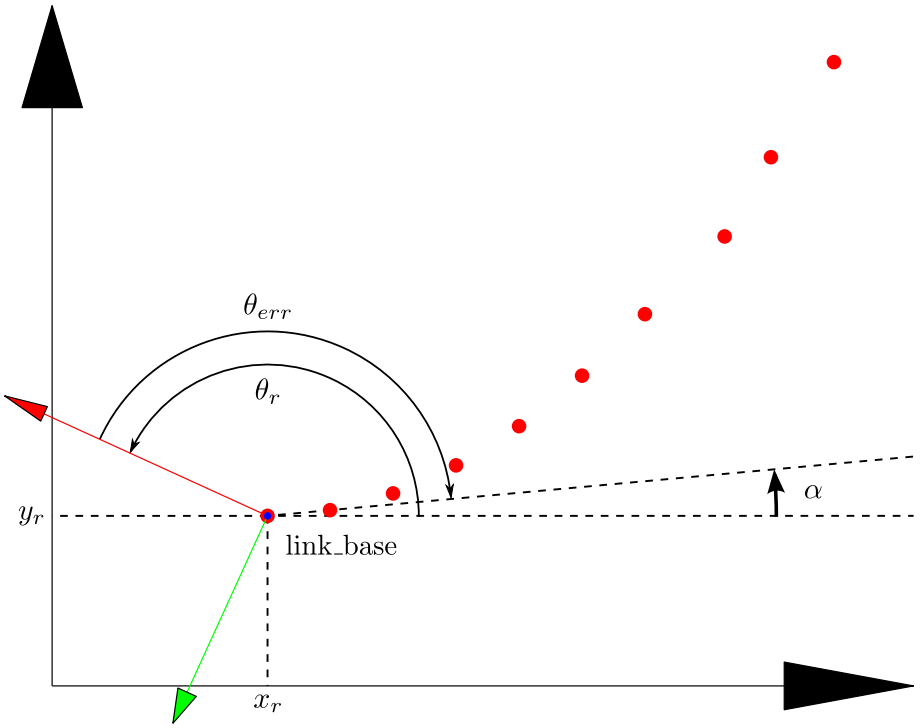

Figura 6.11: Rotación inicial del robot para orientarse hacia la trayectoria.

Para llevar a cabo el movimiento de rotación del robot los motores de las ruedas no se controlan en velocidad, en esta ocasión se controlan en posición. Este tipo de control para los motores de las ruedas es el más adecuado en esta situación ya que se puede calcular cual es el recorrido que deben realizar ambas ruedas para que el robot rote un cierto ángulo. Para realizar

el control en posición de los motores de las ruedas se utilizan los comandos `pvt` de la PMAC. Las siglas `pvt` hacen alusión a posición, velocidad y tiempo, pues estos son parámetros que se le deben comunicar a la PMAC en este comando.

En esta ocasión se han usado comandos `pvt` para controlar los motores de la base móvil en posición en lugar de usar comandos `jog`, del subtipo posición, para reutilizar una rutina escrita en el lenguaje de la PMAC con la que se contaba desde hacía tiempo. Los comandos `pvt` y `jog` no se pueden usar a la vez, son excluyentes. Lo que sí se puede hacer es usar un tipo de comando, parar el motor dirigido por ese comando, y a continuación usar el otro tipo de comando. Es decir, es obligatorio parar el motor antes de cambiar de tipo de comando. Precisamente este comportamiento es el requerido al rotar inicialmente la base para orientarla hacia la trayectoria interés. En primer lugar la base rota controlando los motores en posición con comandos `pvt`. Cuando la base se ha orientado adecuadamente hacia la trayectoria de interés la base se detiene. A partir de entonces los motores de la base pueden comandarse en velocidad con comandos `jog`.

El robot inicialmente se encuentra parado y localizado, a la espera de que el algoritmo de planificación de trayectorias le comunique al algoritmo pure-pursuit cual es la ruta que debe seguir entre su posición actual y un punto destino. Una vez que está disponible la trayectoria se calcula cual la orientación, con respecto al eje x del marco de referencia global, del segmento que une el primer punto de la trayectoria, que es la posición del robot, con el segundo punto de la misma. Se controla que el valor del ángulo α se encuentre entre $[-\pi, \pi]$[2].

$$\alpha = \arctan\left(\frac{y_2 - y_r}{x_2 - x_r}\right) \tag{6.33}$$

Por otro lado se dispone de la orientación inicial del robot, θ_r, proporcionada por el algoritmo del localización global que se está ejecutando en el sistema. Este ángulo también se haya en el intervalo $[-\pi, \pi]$.

El ángulo que debe rotar inicialmente el robot para orientarse hacia la trayectoria es:

$$\theta_{err} = \alpha - \theta_r \tag{6.34}$$

También se controla que este ángulo se encuentre en el intervalo $[-\pi, \pi]$. Por tanto el arco que debe recorrer cada rueda para que el robot rote un ángulo θ_{err} es:

$$arco = \theta_{err} \cdot r_b \tag{6.35}$$

[2]Para asegurar que el valor de α se encuentre en el intervalo $[-\pi, \pi]$ se usa la función `atan2` de la biblioteca estándar de C++.

donde el signo del resultado únicamente indica el sentido en el que la rueda hace el recorrido. Un arco de valor positivo indica que el motor de la rueda ha girado hacia el sentido considerado positivo, mientras que un arco de valor negativo indica que el motor de la rueda ha girado en sentido opuesto al considerado positivo. El ángulo máximo, en valor absoluto, que el robot puede rotar es $\theta_{errmax} = \pi$ rad. También se ha considerado un ángulo mínimo, expresado en valor absoluto, que el robot puede rotar, $\theta_{errmin} = \frac{5 \cdot \pi}{180}$ rad, es decir, 5°. Por tanto, se considera que el robot se encuentra bien orientado hacia la trayectoria si

$$-\frac{5 \cdot \pi}{180} \leq \theta_{err} \leq \frac{5 \cdot \pi}{180}$$

Ahora que ya se sabe la distancia que debe recorrer cada rueda es necesario conocer a que velocidad debe girar cada una de ellas y durante cuanto tiempo deben hacerlo. Para conseguir que el robot rote sobre sí mismo es necesario que ambas ruedas giren en sentidos opuestos a la misma velocidad.

$$v_d(t) = -v_i(t) \tag{6.36}$$

$$\omega_b(t) = \frac{v_d(t) - v_i(t)}{d_b} = \frac{v_d(t)}{r_b} \tag{6.37}$$

$$\text{Giro antihorario: } \theta_{err} \geq 0 \longrightarrow \omega_b(t) \geq 0,\ v_d(t) \geq 0,\ v_i(t) \leq 0 \tag{6.38}$$

$$\text{Giro horario: } \theta_{err} < 0 \longrightarrow \omega_b(t) < 0,\ v_d(t) < 0,\ v_i(t) > 0 \tag{6.39}$$

Los comandos `PVT` también pueden aplicar perfiles S de velocidad a los motores controlados, por lo que se hace uso de esta característica una vez más.

En la figura 6.12 se puede observar un ejemplo de perfil de velocidad del motor de la rueda derecha durante la rotación inicial del robot para orientarse hacia la trayectoria. Durante la fase de rotación inicial los perfiles de velocidad de ambas ruedas tienen signos opuestos y están constituidos por tres tramos. Durante el tramo 1 al motor de cada rueda se le aplica un perfil S de velocidad que le permite pasar de su velocidad inicial, $V_{dI_1} = V_{iI_1} = 0$ cm/s a la velocidad final necesaria para efectuar la rotación, $V_{dF_1} = -V_{iF_1}$. Este primer perfil S de velocidad tiene una duración de $2 \cdot T_1$ ms. Un poco más adelante se detalla el cálculo del valor del término V_{dF_1}. Durante el tramo 2, cada motor mantiene la velocidad final adquirida en el tramo 1.

- $V_{dF_2} = V_{dI_2} = V_{dF_1}$

- $V_{iF_2} = V_{iI_2} = V_{iF_1} = -V_{dF_1}$.

El perfil de velocidad aplicado a los motores durante el tramo 2, aunque no lo parezca también es de tipo S. Un perfil S de velocidad en el que la velocidad final coincide con la velocidad inicial se transforma en un perfil de velocidad constante. La PMAC no tiene ningún problema en trabajar con perfiles S de velocidad en que la velocidad inicial y final sea la misma. Este segundo perfil S de velocidad tiene una duración de $2 \cdot T_2$ ms.

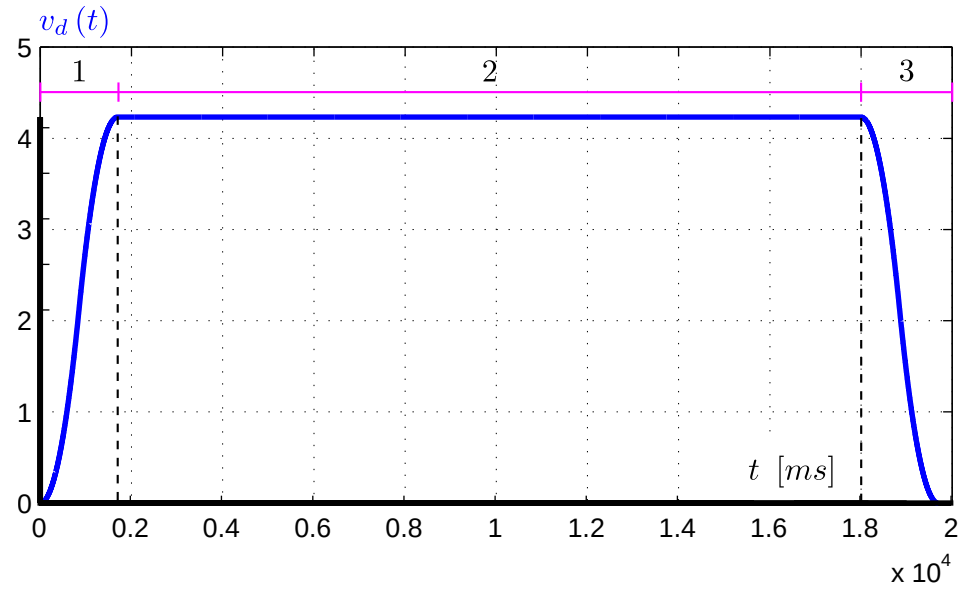

Figura 6.12: Perfil S de velocidad de una de las ruedas de la base durante la rotación inicial de la base.

Por último en el tramo 3 al motor de cada rueda se le aplica de nuevo un perfil S de velocidad que permite que la velocidad de estos pase a ser nula. Este último perfil S de velocidad tiene la misma duración que el primer perfil S, es decir, son perfiles simétricos.

- $V_{dI_3} = V_{dF_2} = V_{dI_2} = V_{dF_1}$, $V_{dF_3} = 0$ cm/s.
- $V_{iI_3} = V_{iF_2} = V_{iI_2} = V_{iF_1} = -V_{dF_1}$, $V_{iF_3} = 0$ cm/s.

Por tanto, el único valor que es necesario calcular es V_{dF_1}, al que se le quita el subíndice 1, quedando únicamente V_{dF}. Resulta lógico pensar que cuanto mayor sea el ángulo que la base tenga que rotar mayor debe ser la velocidad angular a la que ésta rote. Se ha decidido que la

velocidad angular final a la que rote la base sea una función lineal del ángulo de error.

$$W_{bF}\left(\theta_{err}\right) = \begin{cases} \left(\dfrac{W_2 - W_1}{\theta_{errmax} - \theta_{errmin}}\right) \cdot \left(\theta_{err} + \theta_{errmin}\right) - W_1 & si \quad \theta_{err} \leq 0 \\ \left(\dfrac{W_2 - W_1}{\theta_{errmax} - \theta_{errmin}}\right) \cdot \left(\theta_{err} - \theta_{errmin}\right) + W_1 & si \quad \theta_{err} > 0 \end{cases} \tag{6.40}$$

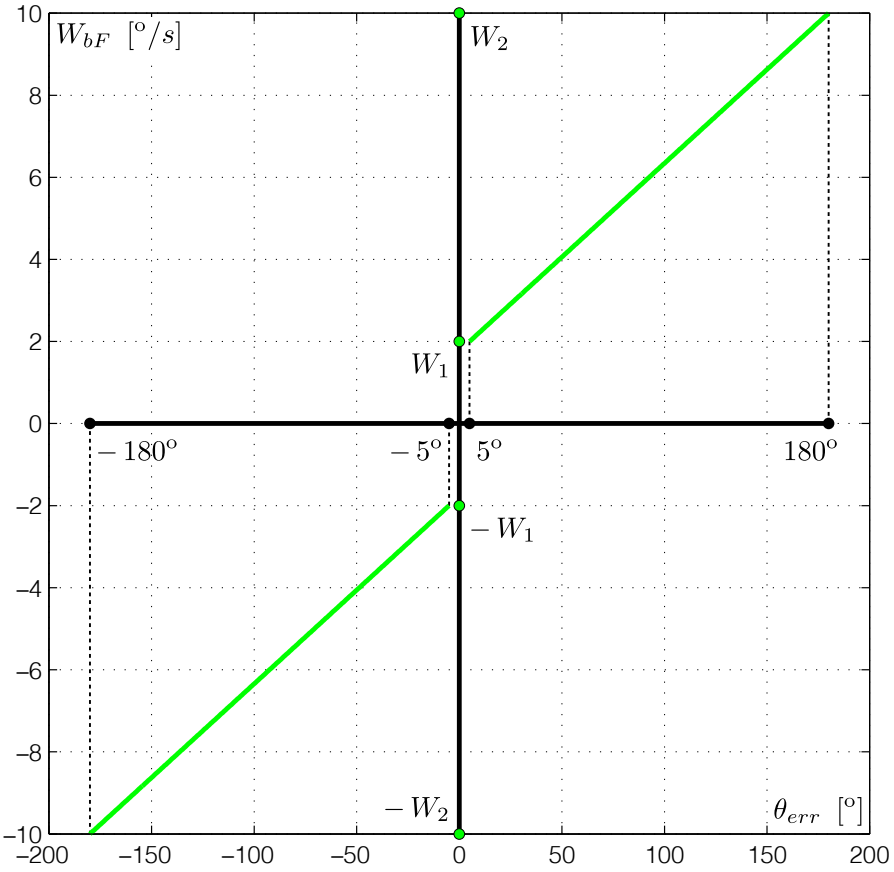

Figura 6.13: Velocidad angular final a la que debe rotar la base como función del ángulo de error.

Para garantizar la estabilidad de la base mientras ésta rota se han usado los siguientes valores:

$$W_2 = \frac{10 \cdot \pi}{180} \text{ rad/s} \tag{6.41}$$

$$W_1 = \frac{2 \cdot \pi}{180} \text{ rad/s} \tag{6.42}$$

Teniendo en cuenta la expresión 6.37, el valor del término V_{dF} se puede expresar como:

$$V_{dF} = W_{bF} \cdot r_b \tag{6.43}$$

En la figura 6.14 se pueden apreciar las funciones V_{dF} y V_{iF} a las que da lugar la función W_{bF}.

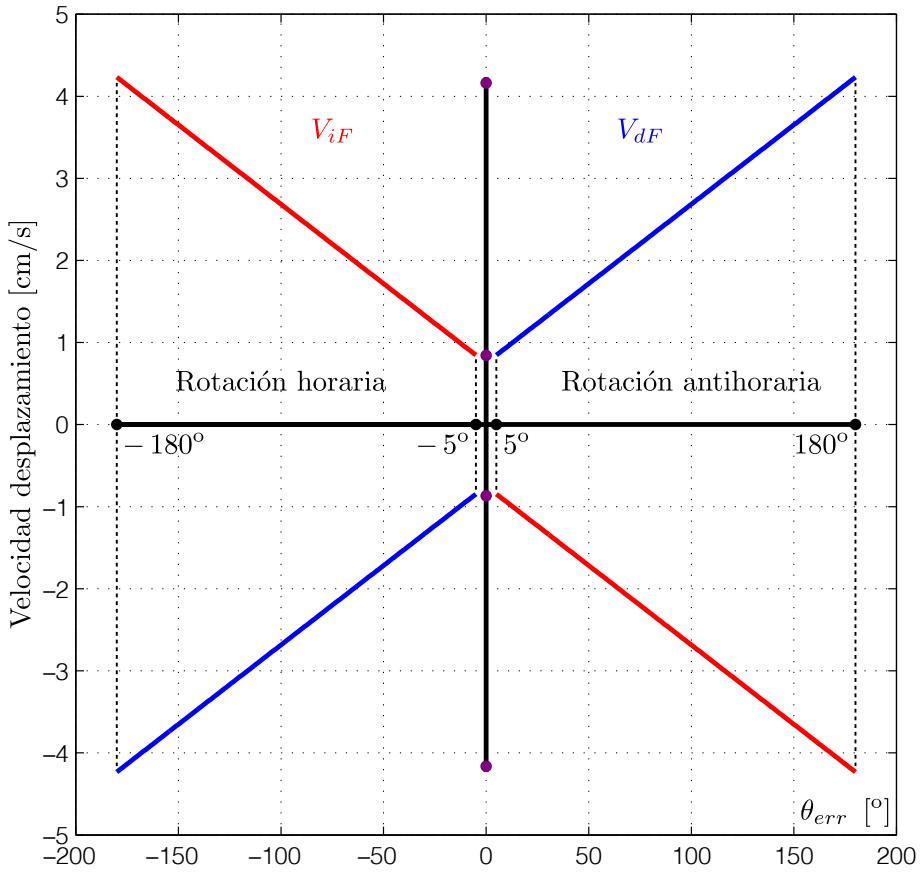

Figura 6.14: Velocidad final de desplazamiento del motor de cada rueda durante la rotación inicial de la base como función del término θ_{err}.

En el movimiento de la rotación la sensación de rapidez con la que se inicia el movimiento de la base es más acusada que la que se tiene cuando el robot inicia el recorrido de una trayectoria normal. Para reducir esa percepción en el observador y aumentar la seguridad del movimiento se ha decidido usar una aceleración máxima, en valor absoluto, inferior a la que se usa durante el recorrido de una trayectoria normal.

$$Acel_{MAX\_rot} = 5\,\text{cm/s}^2 \tag{6.44}$$

Teniendo en el valor del término V_{dF} y del término $Acel_{MAX\_rot}$ el valor del término T_1 es:

$$T_1 = ceil\left(\frac{|\Delta V|}{Acel_{MAX\_rot}}\right) = ceil\left(\frac{|V_{dF}|}{Acel_{MAX\_rot}}\right) \tag{6.45}$$

donde se ha usado de nuevo la función `ceil` para garantizar que el valor de T_1 es un número entero en milisegundos y que se cumple la condición:

$$Acel_{max} = \frac{|V_{dF}|}{ceil\left(\frac{|V_{dF}|}{Acel_{MAX\_rot}}\right)} \leq Acel_{MAX\_rot} \tag{6.46}$$

Tan sólo resta por calcular el valor del término T_2. Del apéndice A se sabe que la distancia recorrida durante un perfil S de velocidad es:

$$\Delta L = T \cdot (V_F + V_I) \tag{6.47}$$

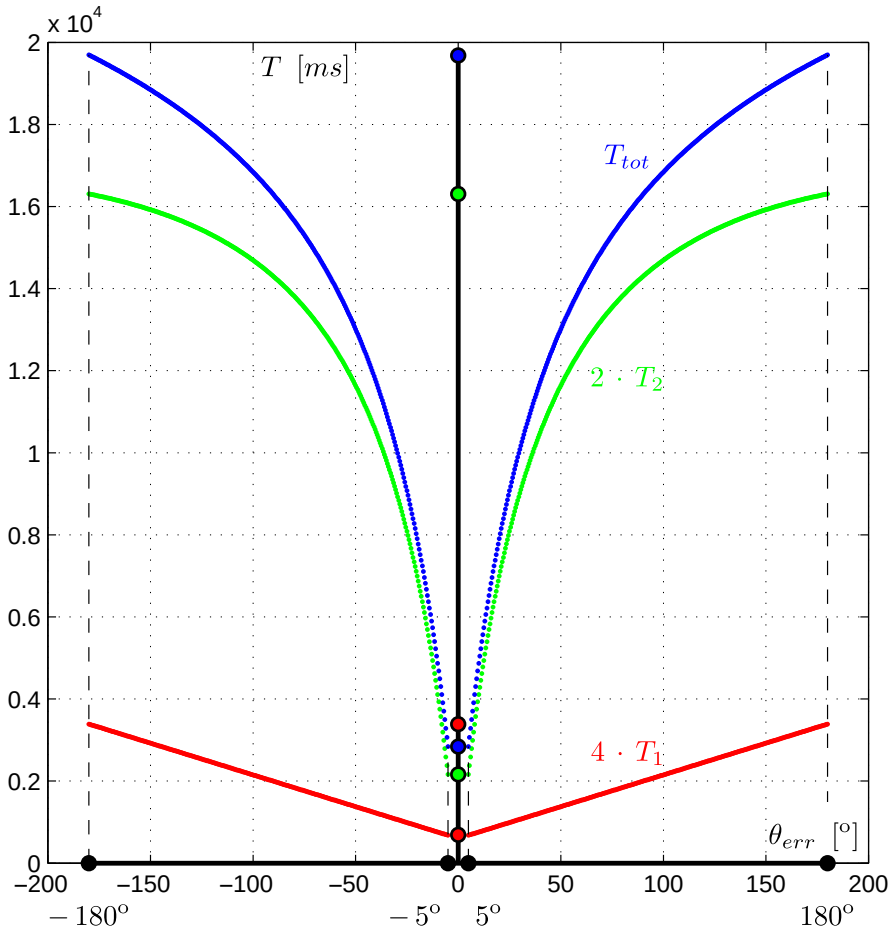

Figura 6.15: Intervalo de tiempo empleado en recorrer el error de orientación θ_{err}.

Por tanto el arco total efectuado por la rueda derecha de la base móvil del robot durante los

tres perfiles S de velocidad es:

$$Arco_{dtot} = Arco_{d1} + Arco_{d2} + Arco_{d3} \tag{6.48}$$

$$= T_1 \cdot (V_{dF_1} + V_{dI_1}) + T_2 \cdot (V_{dF_2} + V_{dI_2}) + T_1 \cdot (V_{dF_3} + V_{dI_3}) \tag{6.49}$$

$$= 2 \cdot V_{dF} \cdot (T_1 + T_2) \tag{6.50}$$

$$= r_b \cdot \theta_{err} = -Arco_{itot} \tag{6.51}$$

Por tanto el valor del término T_2 se puede obtener como:

$$T_2 = ceil\left(\frac{r_b \cdot \theta_{err} - 2 \cdot V_{dF} \cdot T_1}{2 \cdot V_{dF}}\right) \tag{6.52}$$

En esta ocasión se ha usado la función `ceil` sólo para garantizar que el valor del término T_2 es un número entero en milisegundos, pues la aceleración durante el tramo 2 es nula.

Cabe destacar que el tiempo total empleado en recorrer el error de orientación θ_{err} es:

$$T_{tot} = 4 \cdot T_1 + 2 \cdot T_2 \tag{6.53}$$

En la figura 6.15 puede observarse la evolución de los términos $4 \cdot T_1$, $2 \cdot T_2$ y T_{tot}. En la figura 6.16 pueden apreciarse diferentes perfiles de velocidad para el motor de la rueda derecha e izquierda para recorrer diferentes errores de orientación. En la figura 6.17 se puede observar la aceleración aplicada al motor de cada rueda durante el movimiento de rotación inicial de la base.

Una vez que se ha calculado el valor de los términos $Arco_{d1}$, $Arco_{d2}$, V_{dF}, T_1 y T_2 estos son transferidos desde el algoritmo pure-pursuit desarrollado en C++ a la DPRAM. La PMAC recoge estos valores desde un `motion program` y construye tres comandos `pvt`, uno por cada tramo de los anteriormente analizados, que controlan en posición los motores de las ruedas para rotar un ángulo θ_{err}.

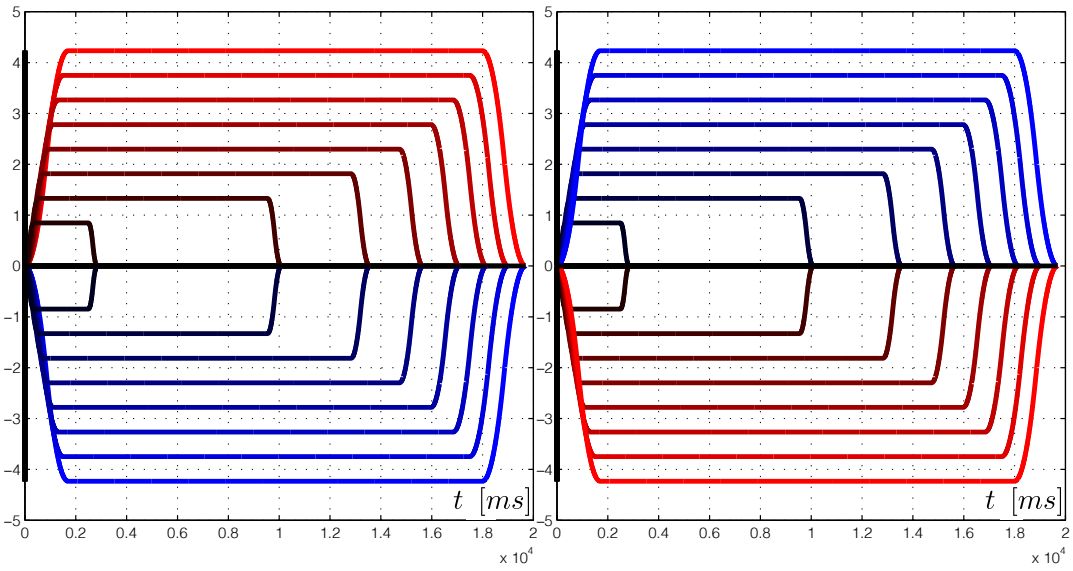

Figura 6.16: Figura izquierda: Perfiles S de velocidad para la rueda izquierda y derecha para rotaciones antihorarias de 5°, 30°, 55°, 80°, 105°, 130°, 155°, 180°. Velocidad expresada en cm/s. Figura derecha: Perfiles S de velocidad para la rueda izquierda y derecha para rotaciones horarias de −5°, −30°, −55°, −80°, −105°, −130°, −155°, −180°. Velocidad expresada en cm/s.

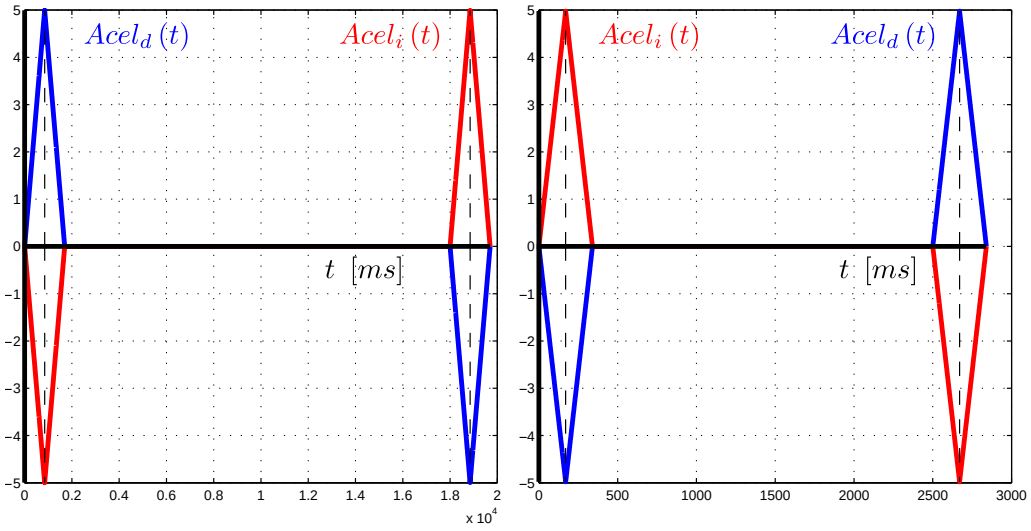

Figura 6.17: Figura izquierda: Aceleración para la rueda izquierda y derecha para una rotación antihoraria de 180°. Figura derecha: Aceleración para la rueda izquierda y derecha para una rotación horaria de −5°.

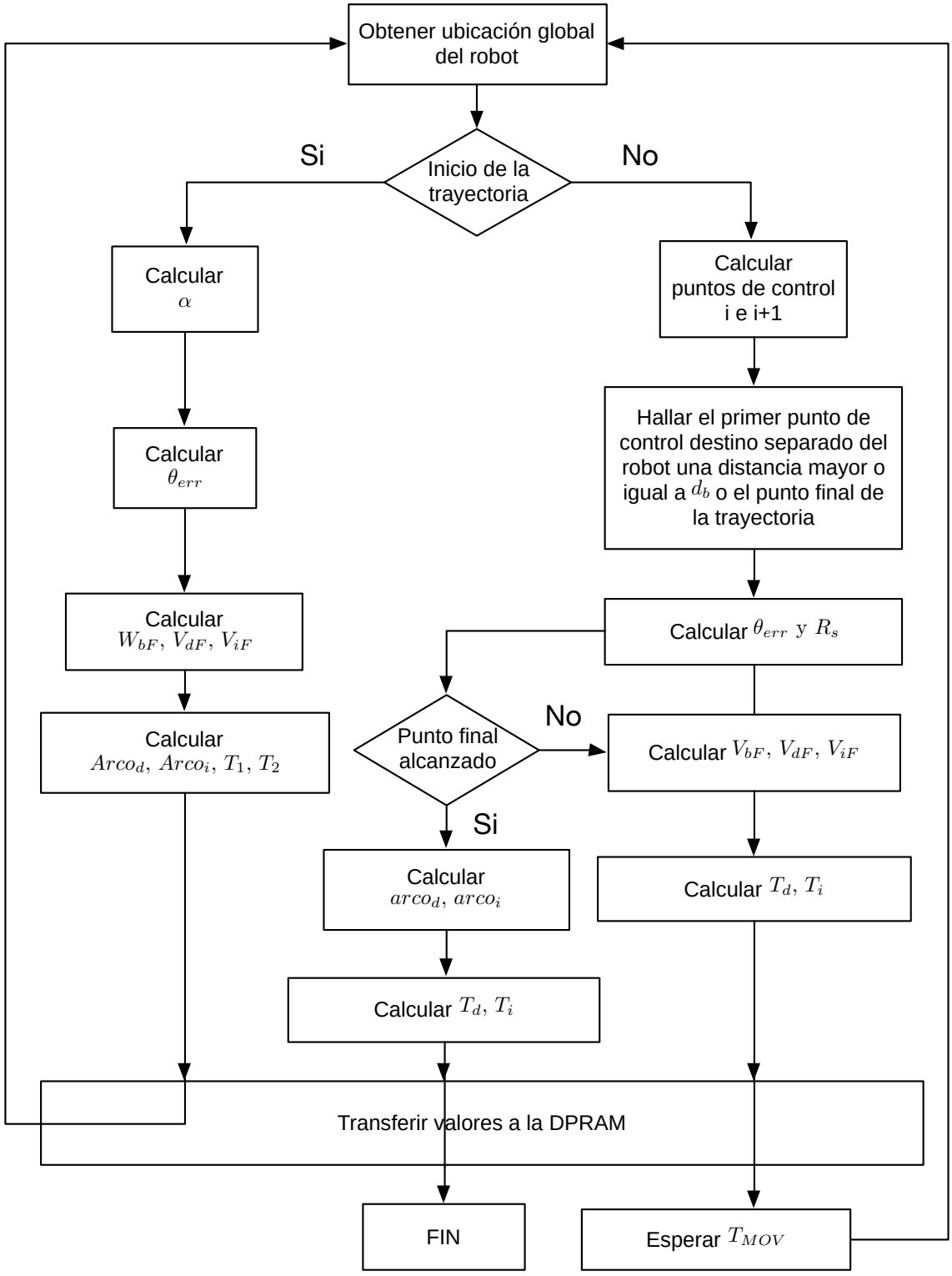

Figura 6.18: Diagrama de flujo del algoritmo pure-pursuit implementado.

Capítulo 7

Resultados, conclusiones y trabajo futuro

En este capítulo se expondrá en primer lugar los resultados obtenidos en la realización de este proyecto. En segundo lugar se describirán las conclusiones obtenidas tras analizar los resultados obtenidos. Finalmente se indicarán algunas líneas de trabajo futuro que pueden ampliar el trabajo realizado en este proyecto.

7.1. Resultados experimentales

Se han realizado pruebas de cada uno de los componentes que forman parte del algoritmo pure-pursuit. En primer lugar se han realizado pruebas del módulo de odometría implementado. Las pruebas han consistido es navegar de forma manual por el Departamento de Sistemas y Automática de la Universidad Carlos III de Madrid. Durante la teleoperación manual del robot se estaba ejecutando el algoritmo de construcción de mapas `g-mapping` presente en el framework `ROS`. Este algoritmo es capaz de construir un mapa del entorno por el que navega el robot usando sólo información obtenida por el telémetro láser e información odométrica relativa al desplazamiento de las ruedas. Si el modelo odométrico sintetizado es correcto el mapa creado debe reflejar adecuadamente el entorno y carecer de distorsiones espaciales.

En la figura 7.1 puede observarse el mapa construido del Departamento de Sistemas y Automática de la Universidad Carlos III de Madrid, donde se encuentra el grupo de trabajo Robotics Lab.

Figura 7.1: Mapa del Departamento de Sistemas y Automática de la Universidad Carlos III de Madrid. En la figura puede apreciarse el recorrido realizado por el robot durante la navegación manual.

Como puede apreciarse el mapa construido refleja fielmente el las instalaciones del Departamento de Sistemas y Automática. Exista una pequeña distorsión que hace que el mapa se encuentre ligeramente inclinado hacia arriba en el flanco izquierdo de la imagen. Este fenómeno es debido a que durante la navegación manual las ruedas del robot se quedaron atascadas en algunos puntos del pasillo principal como consecuencia de algunas irregularidades del suelo. A pesar de dichos sucesos se puede concluir que el mapa es perfectamente usable y por lo tanto que el modelo odométrico construido es correcto.

A modo de ejemplo se ilustra en la figura 7.2 el mapa del laboratorio 1.3.C13 construido también con el algoritmo `g-mapping`.

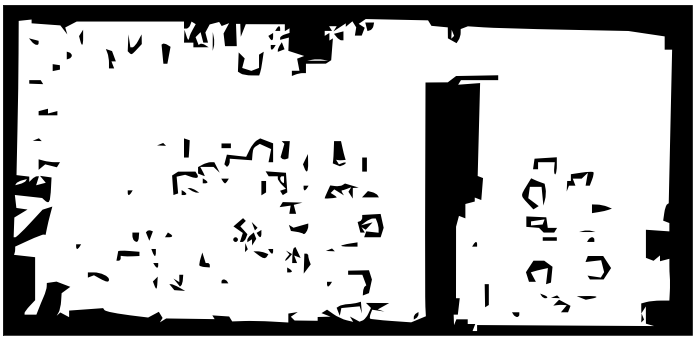

Figura 7.2: Mapa del laboratorio 1.3.C13 de la tercera planta del edificio Agustín de Betancourt de la Universidad Carlos III de Madrid, Leganés.

En segundo lugar se comprobó el correcto funcionamiento del algoritmo de localización global,

basado en un método de optimización evolutivo diferencial. A continuación se describe a grandes rasgos el algoritmo de localización global. Este algoritmo distribuye uniformemente una población inicial de partículas por el mapa del entorno de trabajo. Estas partículas representan ubicaciones posibles del robot en dicho entorno. Desde cada partícula se estiman las N distancias que se obtendrían con el telémetro láser del robot si éste se encontrara físicamente en esa ubicación del entorno. Desde la ubicación física ocupada por el robot en el entorno se obtienen con el telémetro láser las N distancias a los obstáculos que hay a su alrededor. Las N distancias estimadas desde la partícula evaluada y las N distancias obtenidas con el telémetro láser del robot son usadas en una función de fitness, la cual proporciona una número escalar que representa la aptitud de esa partícula para ser la ubicación ocupada por el robot. La partícula con la aptitud más baja es la ubicación estimada del robot. A continuación todas las partículas de la población inicial son sometidas a un proceso de "mezcla". Este proceso está regulado por unos operadores denominados "operador de mutación", "operador de cruce" y "operador de selección". Tras la aplicación de estos operadores se obtiene una nueva población de partículas que son desplazadas de acuerdo a los movimientos del robot. Esta nueva población constituye la población de partida para la siguiente iteración del algoritmo de localización global. Este proceso se repite hasta que todas las partículas se concentran en una circunferencia de $0'5$ m de diámetro, momento en el que se considera que el algoritmo ha convergido y la partícula con menor fitness constituye la ubicación ocupada por el robot. Experimentalmente se ha comprobado que en el mapa de la figura 7.1 el proceso de localización global inicial del robot tarda aproximadamente entre 5 s y 7 s. A continuación se exponen algunas imágenes de diferentes etapas del algoritmo de localización global.

Figura 7.3: Población inicial.

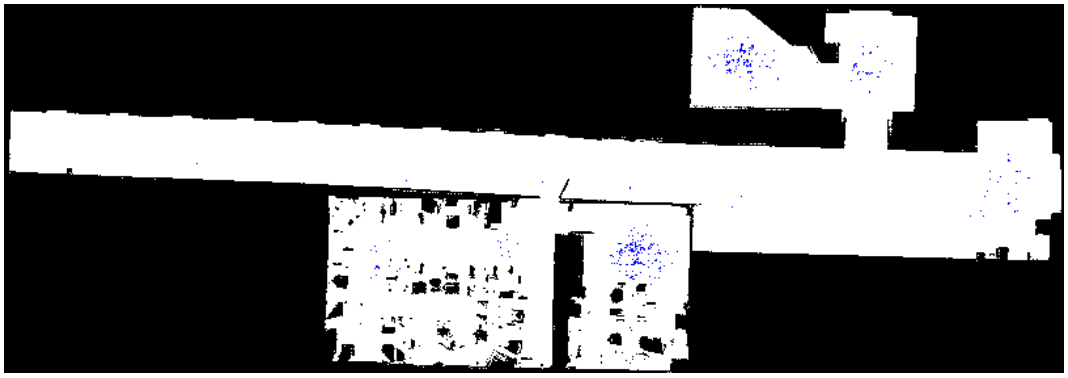

Figura 7.4: Partículas concentradas en torno a las ubicaciones más factibles en las que puede hayarse el robot.

Figura 7.5: Algoritmo de localización a pocas iteraciones antes de su fin.

Figura 7.6: Algoritmo de localización a punto de finalizar.

Figura 7.7: Algoritmo de localización acabado, convergencia conseguida, ubicación del robot encontrada.

Una vez que se ha encontrado la ubicación del robot el problema de localización global se convierte en un problema de relocalización local. Para las relocalizaciones el algoritmo utiliza un número de partículas significativamente menor que el número de partículas usadas durante la localización global. En estas circunstancias las relocalizaciones se completan en tiempos del orden de las centésimas de milisegundos. Para obtener más información acerca de este algoritmo de localización global se sugiere la consulta del documento [1].

Antes de probar el algoritmo pure-pursuit en la plataforma real se ha probado en un programa simulador desarrollado expresamente para comprobar la eficacia del algoritmo y averiguar el valor adecuado de los parámetros de control del mismo. Este simulador ha sido elaborado en el lenguaje de programación de la aplicación Matlab y para ello ha sido necesario desarrollar el mismo algoritmo pure-pursuit que el que ha sido desarrollado en C++. Ha sido necesario incorporar al simulador el control en velocidad y en posición que es capaz de realizar la PMAC sobre los motores, es decir, ha habido que sintetizar las ecuaciones que generan los perfiles S de velocidad que se aplican a los motores de la base, las ecuaciones que proporcionan las aceleraciones que experimentan los motores de la base y las ecuaciones que proporcionan los recorridos efectuados por los motores de la base. También ha sido necesario implementar un método de integración numérica (la regla del trapecio) con el objetivo de estimar la ubicación del robot en cualquier instante de tiempo a partir de las velocidades de ambas ruedas. La PMAC proporciona el valor de la velocidad en un motor cada milisegundo, de ahí que en el simulador se haya usado el tiempo expresado en milisegundos en todas aquellas expresiones que hagan uso de esta magnitud. Una resolución temporal de un milisegundo es excelente, de ahí que a pesar de usar una regla de

integración numérica tan sencilla se obtengan resultados estimados muy exactos.

Al explicar el algoritmo pure-pursuit en el capítulo 6 no se ha tenido en cuenta el hecho de que Manfred no es un robot simétrico en el plano sagital. La decisión de explicar el algoritmo de este modo se ha tomado adrede con el objetivo de hacer menos complicada de lo estrictamente necesario su descripción. Todas las expresiones y razonamientos explicados en el capítulo 6 son perfectamente válidos para un robot simétrico en el plano sagital, es decir, si Manfred no tuviera ningún brazo o poseyera dos, uno a cada lado de su torso. Sin embargo Manfred posee sólo un brazo, situado en el flanco derecho de su torso. Este brazo sobresale 10 cm del extremo derecho de la base móvil, lo cual provoca que la anchura total del robot no sea la anchura de la base, 55 cm, sino la anchura de ésta más la prolongación del brazo fuera de ella, 65 cm.

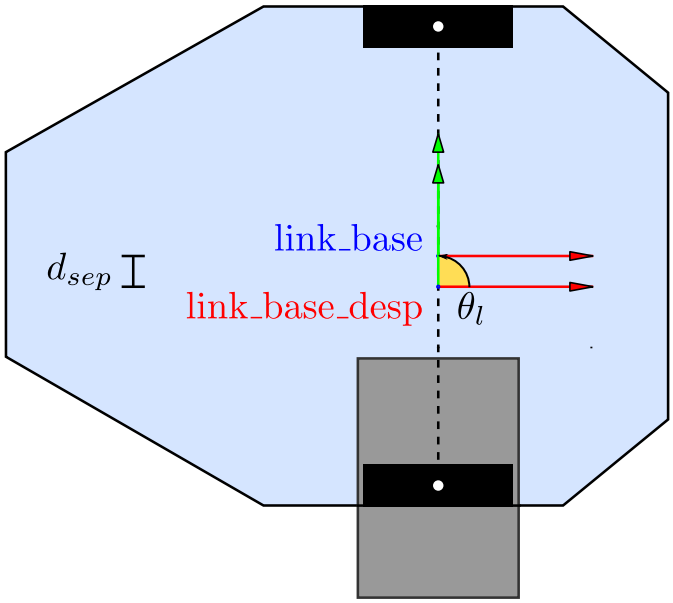

Figura 7.8: Relación entre los marcos de referencia `link_base` y `link_base_desp`.

Por este motivo el sistema de referencia que se había considerado maestro o principal durante la explicación del algoritmo en el capítulo 6, `link_base`, ya no tiene esta categoría. Teniendo en cuenta la asimetría sagital de Manfred el nuevo sistema de referencia maestro es `link_base_desp`. Este sistema de referencia se encuentra en la mitad de la anchura total del robot, desplazado $d_{sep} = 5$ cm respecto al eje y negativo del marco de referencia `link_base`. El sistema de referencia `link_base_desp` se sitúa, de nuevo, sobre la línea imaginaría que une los centros de ambas ruedas, a mitad de la altura de la base. La adopción del sistema de referencia `link_base_desp`

como marco de referencia maestro tiene varias consecuencias. En primer, dado que el sistema de referencia `link_base_desp` está situado en la mitad de la anchura del robot es éste marco de referencia el que tiene que pasar sobre los puntos de control de la trayectoria de interés. Por tanto durante este capítulo se ha coloreado en rojo la trayectoria de interés generada por el planificador. Esta trayectoria es la que debe seguir el marco de referencia `link_base_desp`. En azul se muestra la trayectoria de interés que seguiría el marco de referencia `link_base`. La trayectoria de interés azul sólo se muestra con fines didácticos, en realidad no se usa para nada y no es necesario calcularla a partir de la trayectoria de interés de color rojo. En la figura 7.9 se puede apreciar en rojo una trayectoria de interés de prueba usada en el simulador. En azul se muestra la trayectoria que debería seguir el marco de referencia `link_base`. En magenta se muestra un fragmento de ruta trazada por el marco de referencia `link_base_desp`. En azul cyan se muestra una fragmento de ruta trazada por el marco de referencia `link_base`.

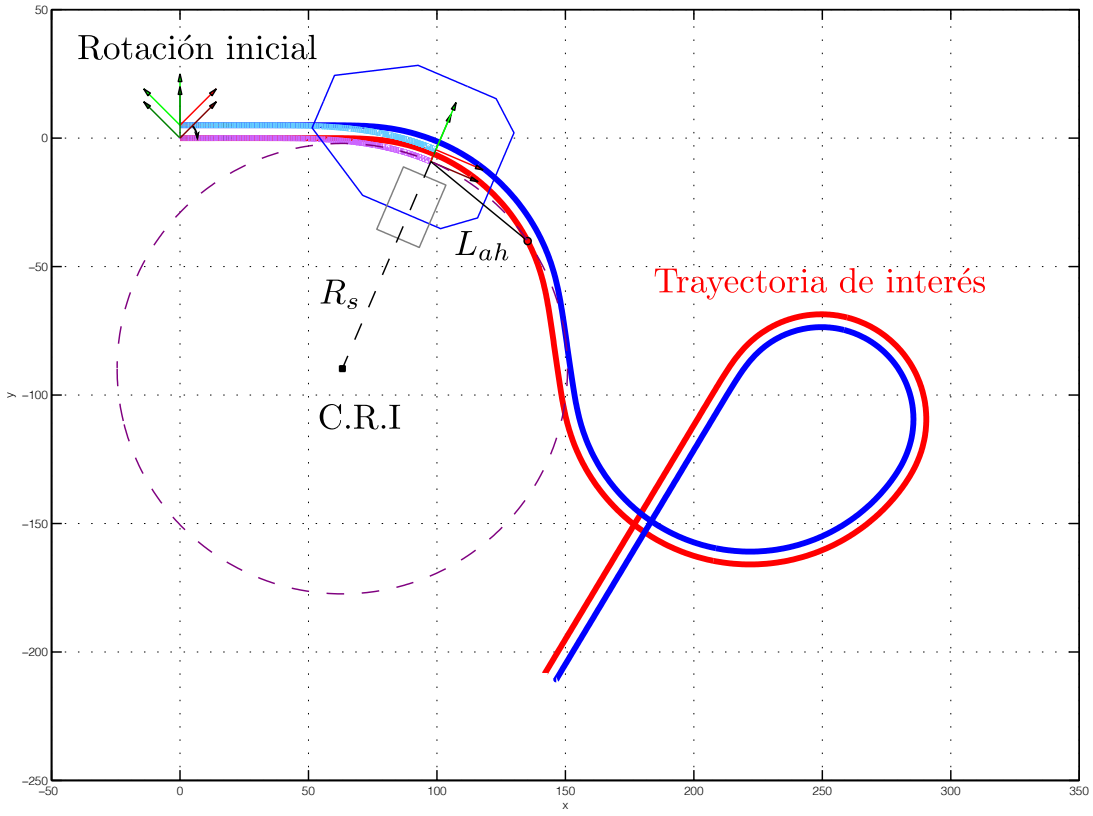

Figura 7.9: Trayectoria de prueba usada en el programa de simulación. Coordenadas x e y expresadas en cm.

En el capítulo 6 se mostró que las velocidades finales de desplazamiento lineal de las ruedas de la base móvil que permiten trazar un arco cuyo radio de giro es R_s se obtienen con las expresiones:

$$V_{dF} = \left(1 + \frac{r_b}{R_s}\right) \cdot V_{bF}\left(|R_s|\right)$$
$$V_{iF} = \left(1 - \frac{r_b}{R_s}\right) \cdot V_{bF}\left(|R_s|\right)$$

En estas expresiones el término R_s representa la distancia que separa el punto medio de la línea imaginaria que une los centro de ambas ruedas del C.R.I. Sin embargo, ahora que consideramos la asimetría sagital de Manfred, el término R_s representa la distancia entre el marco de referencia `link_base_desp` y el C.R.I. Por este motivo para averiguar la distancia que separa el punto medio de la línea imaginaria que une los centro de ambas ruedas del C.R.I, es decir, el radio de giro efectivo, se usa la corrección:

$$R_{sefec} = R_s - d_{sep} \tag{7.1}$$

Así en todas las expresiones vistas en el capítulo 6 donde se usó el término R_s ahora se debe usar R_{sefec}. De este modo tan sencillo se puede hacer que el marco de referencia `link_base_desp` siga una trayectoria calculada por el planificador.

Dos aspectos más a tener en cuenta. El algoritmo de localización global con evolución diferencial proporciona la ubicación del marco de referencia `link_base` en el marco de referencia global. Por tanto para conocer la ubicación del sistema de referencia `link_base_desp` durante la ejecución del algoritmo pure-puruist se debe usar la biblioteca `tf` de `ROS`, la cual utiliza el árbol de transformaciones de Manfred, definido en un fichero `urdf`. Ver capítulo 5. En segundo lugar la odometría proporciona la ubicación del marco de referencia `link_base` en el sistema de referencia odométrico, es decir, un sistema de referencia que ocupa la misma ubicación que la que tenía el marco de referencia `link_base` en el momento de activar el módulo odométrico. Tras estas aclaraciones se da paso a la explicación del resto de pruebas llevadas a cabo.

En la siguiente prueba se ha ejecutado el algoritmo pure-pursuit en el simulador, se ha obtenido la ruta que realiza el marco de referencia `link_base_desp` y después se ha ejecutado en el robot real el algoritmo pure-pursuit, registrando la trayectoria seguida únicamente mediante odometría. Después la trayectoria seguida en el simulador se compara con la trayectoria seguida por el sistema robótico.

En la figura 7.9 se puede apreciar que el robot virtual, en su ubicación inicial, no estaba orientado hacia la trayectoria, de ahí que haya tenido que realizar un primer movimiendo de

rotación horaria. Una vez que el robot virtual estaba alineado con la trayectoria ha comenzado a ejecutar el algoritmo pure-pursuit. En la figura anterior se puede apreciar una ubicación ocupada por el robot virtual, el punto destino seleccionado en la trayectoria, separado del robot virtual una distancia $L_{ah} \geq d_b$, y el radio de giro R_s que permite trazar un arco de circunferencia que conecta ambas posiciones.

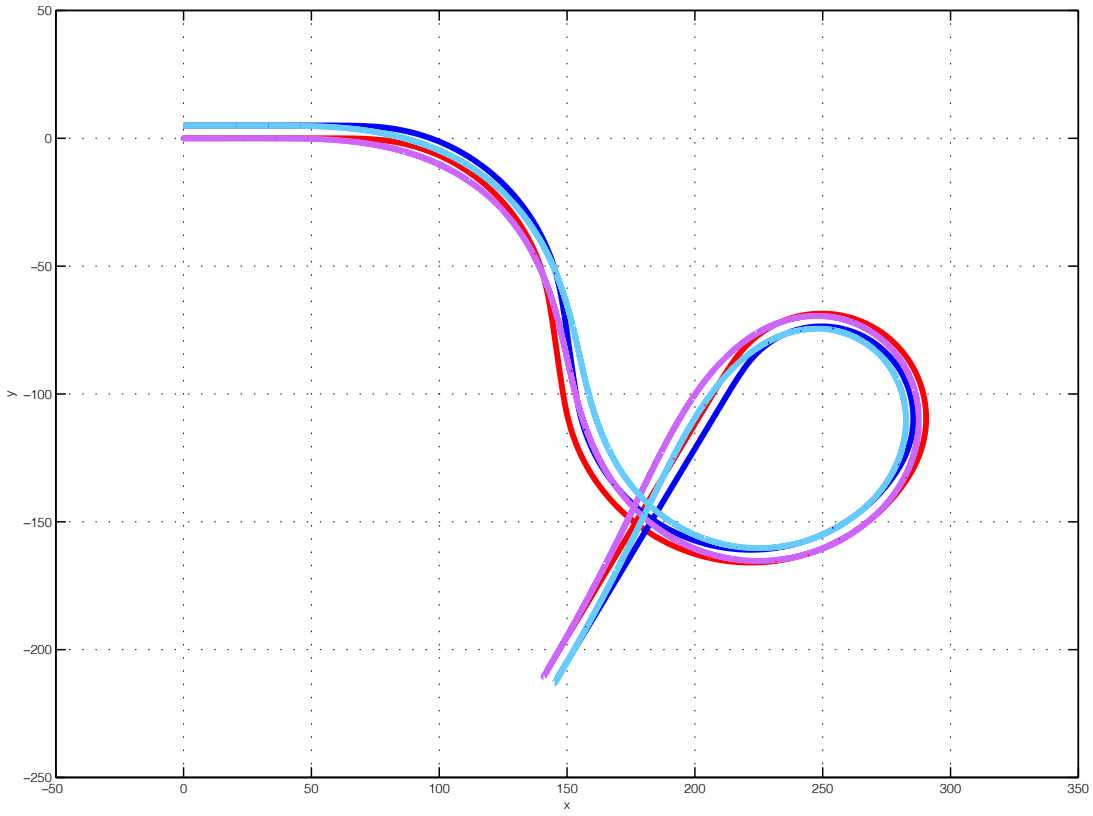

Figura 7.10: Trayectoria completada en el simulador.

En la figura 7.10 se puede observar la trayectoria completada en el simulador. Obviamente la trayectoria seguida concuerda perfectamente con la trayectoria de interés. En la figura 7.10 puede apreciarse el efecto característico de suavizado de trayectoria que provoca el algoritmo pure-pursuit, principalmente alrededor de los tramos curvos. Este fenómeno se debe al valor de la distancia L_{ah} usado. Cuanto menor es el valor del término L_{ah} más se ciñe la trayectoria seguida a la trayectoria de interés y cuanto mayor es el valor de este término menos se ciñe la trayectoria seguida a la trayectoria de interés, en otras palabras, mayor es el suavizado de la trayectoria seguida.

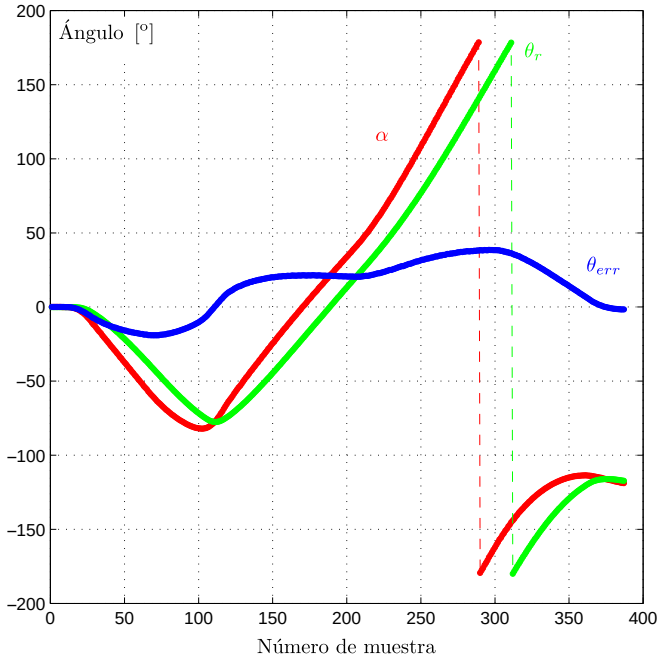

Figura 7.11: Evolución del ángulo de error θ_{err}.

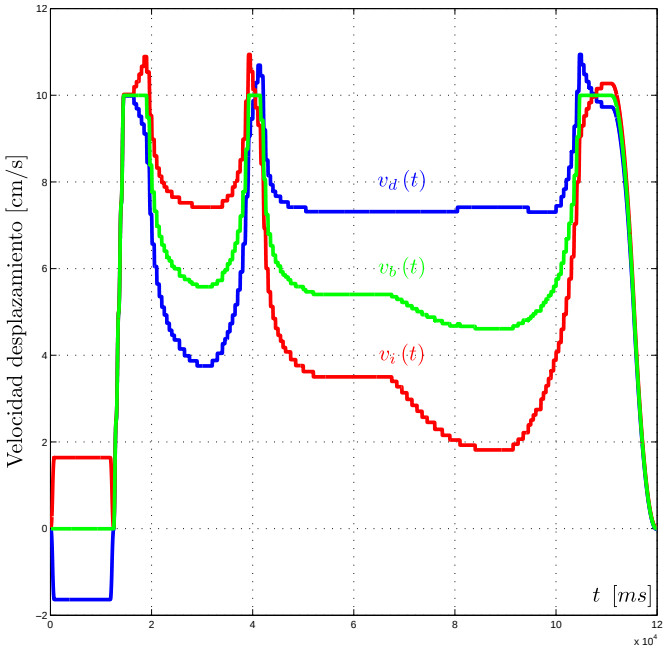

Figura 7.12: Velocidad lineal de desplazamiento de cada rueda de la base y velocidad lineal de desplazamiento del robot.

En el simulador también se puede visualizar la evolución del ángulo formado por la horizontal y el segmento que une la posición del sistema de referencia `link_base_desp` con la posición seleccionada en la trayectoria de interés, α, la evolución de la orientación del robot, θ_r, y la evolución del ángulo de error. θ_{err}. Ver figura 7.11.

Como puede observarse en la figura 7.11 la orientación del robot, θ_r, se haya siempre siguiendo de cerca al ángulo α, como era de esperar, ya que este es el objetivo del algoritmo pure-pursuit, reducir el ángulo de error, θ_{err}, para mantener al robot orientado hacia la trayectoria. En los tramos curvos el error de orientación aumenta. Cuando mayor es el radio de giro de la curva a la que se enfrenta el robot mayor es el valor del error de orientación. Todos los ángulos considerados en el algoritmo pure-pursuit se hayan en el intervalo $[-\pi, \pi]$, por ese motivo en las cercanías de la muestra 300 de la figura 7.11 se produce ese cambio tan brusco de α y θ_r, ya que cualquier ángulo que exceda los π rad se encontrará en las inmediaciones de $-\pi$ rad, habiéndose producido un salto de 2π rad.

Entre los muchos datos que proporciona el simulador se encuentran los perfiles de velocidad de desplazamiento lineal aplicados a los motores de la base durante todo el tiempo que dura el trayecto, la velocidad lineal de desplazamiento de la base y la velocidad angular de la base. Ver figura 7.12.

Analizando estos perfiles de velocidad se puede conocer cómo se ha desplazado el robot durante el recorrido de la trayectoria de interés. En la figura 7.13 se presentan las velocidades de desplazamiento de ambas ruedas de la base normalizadas, la velocidad lineal de desplazamiento de la base normalizada y la velocidad angular de la base normalizada. El proceso de normalización permite que estas funciones, que tienen rangos de valores diferentes, puedan ser comparadas. Entre el origen de coordenadas de la figura 7.13 y el punto 1 se observa que las velocidades de ambas ruedas tienen la misma magnitud y sentido opuesto, de este modo el robot está rotando para orientarse hacia la trayectoria. Como la velocidad de la rueda derecha es negativa el sentido de rotación es horario. También se puede apreciar el sentido de rotación en la velocidad angular, negativa, por tanto rotación horaria. Precisamente por tener las velocidades de la base misma magnitud y sentido opuesto la velocidad lineal de desplazamiento de la base es nula. Entre el punto 1 y el punto 2 la velocidad angular de la base es nula, lo que denota que el robot se ha desplazado en línea recta, siempre de frente. Este fenómeno también se puede comprobar observando que la velocidad lineal de desplazamiento de la rueda derecha, de la rueda izquierda y del robot coinciden pues los trazos de estas funciones se superponen, siendo visible únicamente

el de la función $v_b\,(t)$.

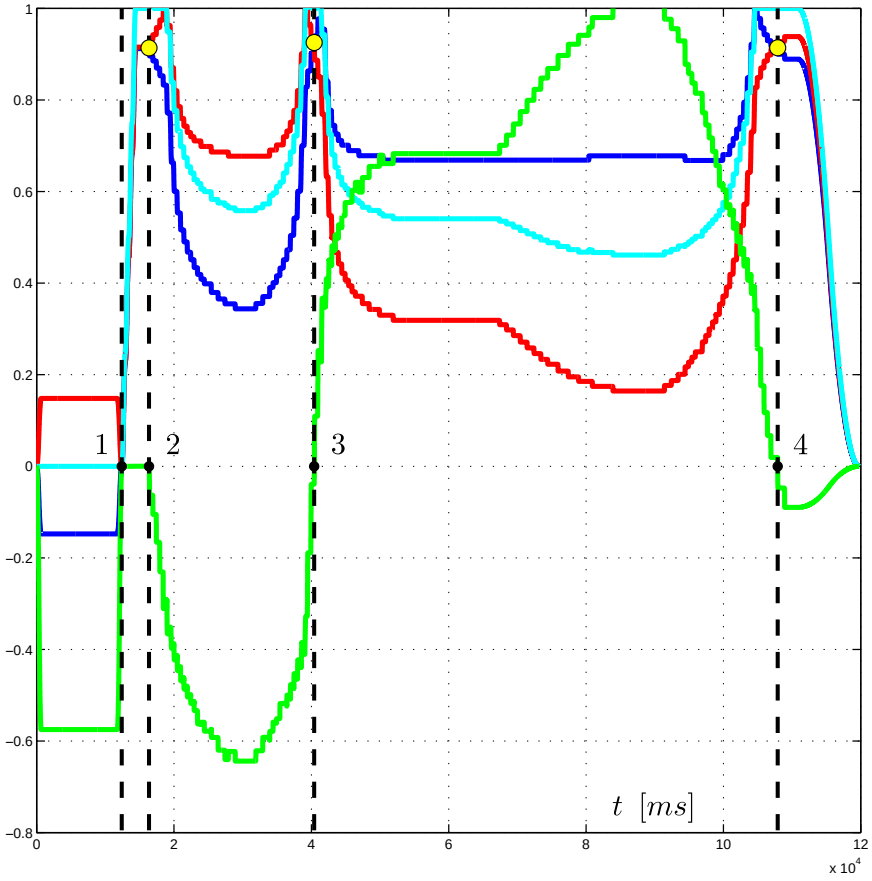

Figura 7.13: Velocidad de desplazamiento lineal de cada rueda normalizada, velocidad de desplazamiento lineal de la base normalizada y velocidad angular de la base normalizada.

Entre el punto 2 y el punto 3 la velocidad lineal de desplazamiento de la rueda izquierda es mayor que la velocidad lineal de desplazamiento de la rueda derecha por lo que el robot describe en todo este tiempo un movimiento de traslación circular horario. En el punto 3 se observa que la velocidad angular se hace nula, lo que indica que por un instante la velocidad lineal de desplazamiento de la rueda derecha e izquierda coinciden. En este punto se produce un cambio en el sentido de giro del robot. A partir del punto 3, y a hasta el punto 4, el robot describe un movimiento de traslación circular antihorario, como se deduce al observar que la velocidad lineal de desplazamiento de la rueda derecha es superior a la velocidad lineal de desplazamiento de la rueda izquierda. Finalmente en el punto 4 el robot corrige un poco su trayectoria, realizando un breve movimiento de traslación circular horario, para a continuación efectuar un último tramo

casi recto, denotado por una velocidad angular próxima a cero y unas velocidades lineales de desplazamiento de la rueda derecha, izquierda y de la base que se solapan.

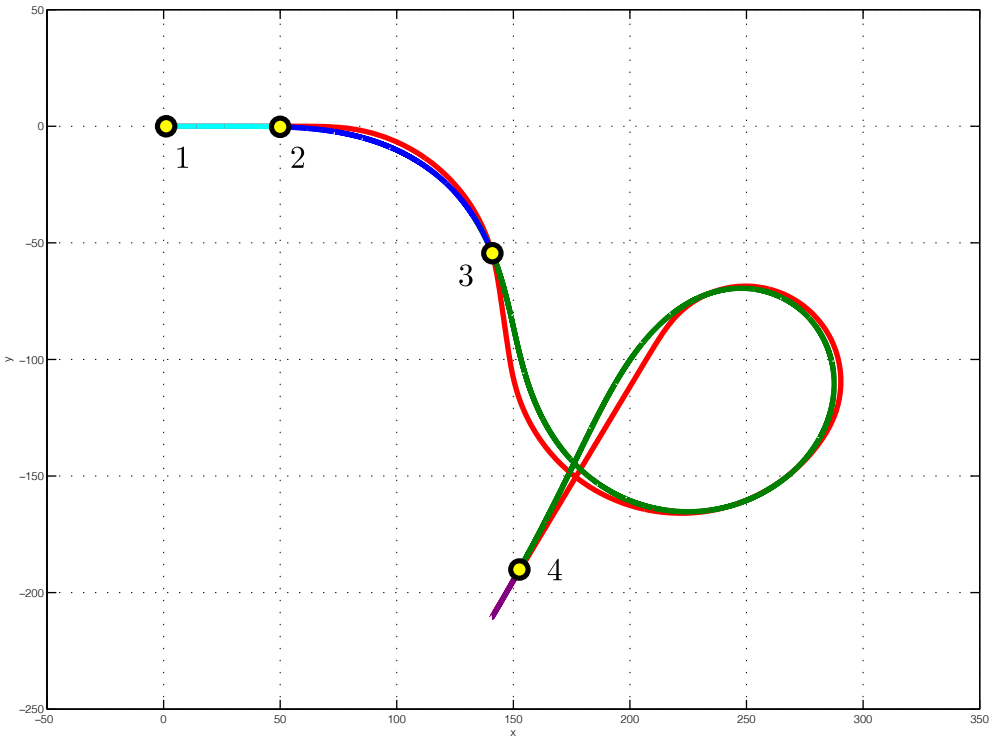

Figura 7.14: Identificación de tramos explicados en la figura 7.13 sobre la trayectoria seguida del robot.

A continuación se ha probado el algoritmo pure-pursuit, desarrollado en C++, en el sistema robótico real con la trayectoria en forma de lazo usada en el simulador. Esta trayectoria no ha sido creada con el planificador de trayectorias que está instalado en el robot, se trata de una trayectoria artificial creada off-line expresamente para esta prueba. La prueba ha tenido lugar en el laboratorio 1.3.C13, en un entorno libre de obstáculos. La segunda salvedad con respecto al funcionamiento normal del sistema es que en esta ocasión no se ha usado el módulo de localización global con evolución diferencial para conocer la ubicación del robot en el entorno. Para localizarse globalmente el robot ha contado únicamente con la odometría calculada según las expresiones del capítulo 4, en base el movimiento de los motores de la base. La odometría es capaz de proporcionar la ubicación global del robot en el entorno siempre y cuando se le indique una ubicación global de partida. La ubicación global inicial en la que se encontraba el robot y que se ha comunicado como punto de partida al módulo odométrico es $\left(x_r = 0, y_r = 0, \theta_r = \frac{\pi}{4}\right)$.

El objetivo de esta prueba es comprobar si el algoritmo pure-pursuit es capaz de funcionar adecuadamente únicamente haciendo uso de la información odométrica, como medida temporal de precaución en caso de que la localización global con evolución diferencial dejase de estar disponible durante un lapso breve de tiempo debido a que no se encuentre la ubicación estimada del robot.

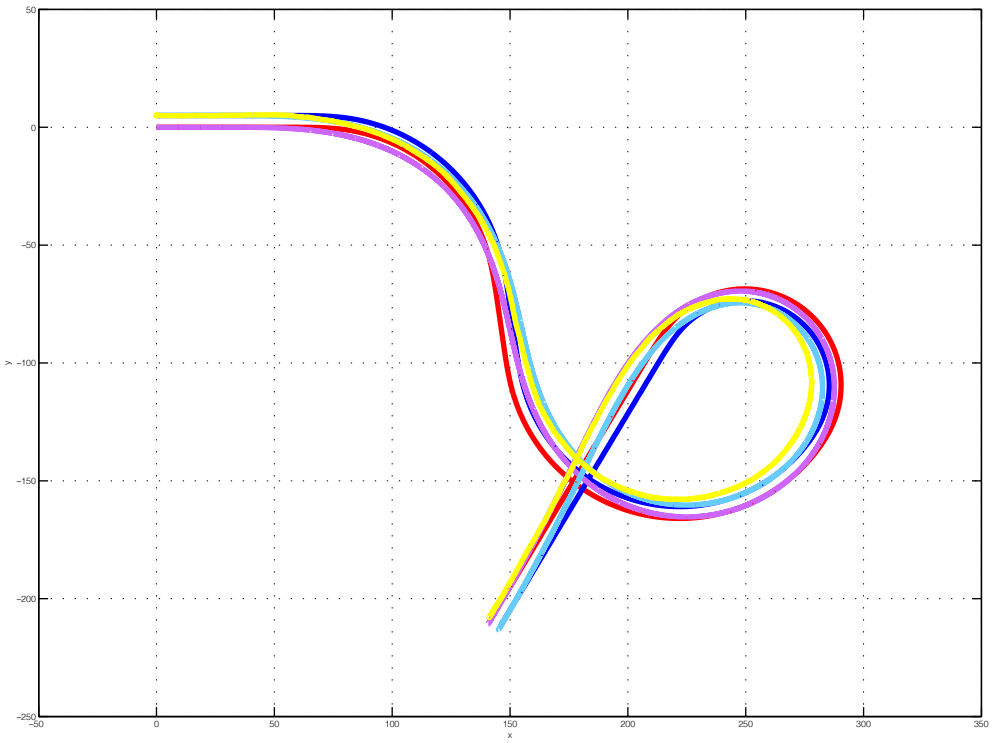

Figura 7.15: Compartiva de trayectorias. En rojo la trayectoria de interés que debe seguir el marco de referencia `link_base_desp`. En azul la trayectoria de interés que debe seguir el marco de referencia `link_base`. En magenta la ruta trazada por el sistema de referencia `link_base_desp` en el simulador. En azul cyan la ruta trazada por el sistema de referencia `link_base` en el simulador. En amarillo la trayectoria seguida por el marco de referencia `link_base_desp` obtenida a partir de la información proporcionada por el módulo odométrico

En la figura 7.15 se puede apreciar la trayectoria original, en rojo, la trayectoria seguida por el sistema de referencia `link_base_desp` en el simulador, en magenta, y la trayectoria del sistema de referencia `link_base_desp` registrada por la odometría, en amarillo. Según el módulo odométrico el marco de referencia `link_base_desp` no finaliza en el mismo punto que la trayectoria original. En el laboratorio se pudo comprobar que el punto medio de la línea imaginaria que une el centro

de ambas ruedas, es decir, el marco de referencia `link_base`, finalizaba "aproximadamente" sobre punto final de la trayectoria de interés. Se dice "aproximadamente" ya que las dimensiones de la base impidieron comprobar si la coincidencia de ambos puntos era perfecta. En cualquier caso, si la coincidencia de ambos puntos no era perfecta la desviación entre ellos era a lo sumo de unos pocos centímetros en cada eje, es decir, estaríamos hablando de un cuadrado de incertidumbre de 5x5 cm centrado en el punto final de la trayectoria de interés, lo cual resulta más que aceptable tratándose de una navegación que ha usado únicamente información odométrica como medio de localización.

Debido al resultado de la prueba anterior se puede afirmar que el algoritmo pure-pursuit puede seguir una trayectoria temporalmente usando localización global basada en odometría ante una incapacidad breve del algoritmo de localización global por evolución diferencial de encontrar la ubicación estimada del robot en el entorno.

La última prueba que se va a explicar hace uso de todos los módulos con los que cuenta el robot. La prueba consiste en hacer que el robot, situado en un lugar desconocido en una de la dos salas que constituyen el laboratorio 1.3.C13 del Departamento de Sistemas y Automática de la Universidad Carlos III de Madrid, siga una trayectoria que comunica su posición con un punto destino elegido en la sala adyacente. La sala en la que se encuentra el robot y el punto destino elegido, proporcionado explícitamente al algoritmo pure-pursuit al arrancarlo, harán que el planificador proporcione una trayectoria que necesariamente debe pasar por el hueco de la puerta que comunica ambas salas y cuya anchura es poco mayor que la anchura del robot. En la anchura del robot se tiene en cuenta la parte del brazo que sale fuera del lateral derecho de la base. Por tanto el algoritmo pure-pursuit desarrollado tiene que conseguir que el robot siga la trayectoria de interés con suma exactitud para evitar su colisión con el marco de la puerta.

La ubicación global del robot, proporcionada por el algoritmo de localización global con evolución diferencial, es publicada mediante el framework `ROS`. Esto significa que todos aquellos nodos que se suscriban al `topic` adecuado recibirán la ubicación global del sistema de referencia `link_base` en el mapa del entorno. Dado que Manfred está conectado a una red inalámbrica de laboratorio los mensajes publicados por cualquiera de sus nodos pueden ser recibidos por otros nodos que se estén ejecutando en otras máquinas distintas a la del propio robot, siempre que estas máquinas estén conectadas a la misma red inalámbrica que el robot. En una máquina del laboratorio 1.3.C13 se ha lanzado el nodo de visualización de datos del framework `ROS`, el nodo `rviz`. Este nodo permite visualizar todo tipo de información que se gestione a través de `ROS`.

En este caso el nodo `rviz` se suscribe al topic que transporta la ubicación global del marco de referencia `link_base` y representa el modelo `urdf` de Manfred en dicha ubicación en el mapa del entorno. De este modo cualquiera que disponga de un PC conectado a la red inalámbrica del laboratorio puede observar el recorrido realizado por el robot. Parte de la información visualizada en el nodo `rviz` se muestra en las imágenes 1 a 8 de las figuras 7.16 y 7.17.

El algoritmo de localización global con evolución diferencial tarda entre 5 y 7 segundos en encontrar la ubicación del robot en el mapa del entorno. Imagen 1 de la figura 7.16. A continuación el algoritmo de planificación calcula la ruta óptima entre la posición del marco de referencia `link_base_desp` y un punto destino que se le ha indicado al algoritmo expresamente, en coordenadas globales. Esta ruta tiene la particularidad de que atraviesa la puerta del laboratorio 1.3.C13, cuya anchura es poco mayor que la anchura total del robot, por lo tanto el algoritmo pure-pursuit debe seguir la trayectoria de interés con suma exactitud. En la figura 2 de la imagen 7.16 se pueden apreciar dos rutas, una de color rojo y otra de color azul. La trayectoria proporcionada por el algoritmo de planificación es la de color rojo. Esta ruta es la que debe seguir el marco de referencia `link_base_desp`. En esta ocasión no es necesario una rotación inicial del robot para orientarse hacia la trayectoria, ya que el error de orientación entre la trayectoria roja y la orientación del marco de referencia `link_base_desp` es inferior a $|\pm 5°|$, que es el límite mínimo a partir del cual rotar.

Durante todo el tiempo que dura el recorrido de la trayectoria de interés el algoritmo pure-pursuit realiza las siguientes tareas:

1. Obtener la ubicación absoluta del marco de referencia `link_base` del algoritmo de localización global con evolución diferencial.

2. Usar la librería `tf` del framework `ROS` para obtener la ubicación global del sistema de referencia `link_base_desp` a partir de la ubicación global del marco de referencia `link_base`.

3. Obtener los puntos de control i e $i + 1$ de la trayectoria de interés roja entre los que se encuentra la proyección ortogonal de la posición del marco de referencia `link_base_desp`.

4. Encontrar el primer punto de la trayectoria de interés roja, empezando la búsqueda en el punto $i + 1$, que se encuentra a una distancia mayor o igual al diámetro de la base, d_b, del marco de referencia `link_base_desp`.

5. Calcular el error de orientación, θ_{err}, entre la orientación del robot, θ_r, y la orientación

del segmento de recta que une la posición del marco `link_base_desp` con el punto destino hallado en el paso anterior, α.

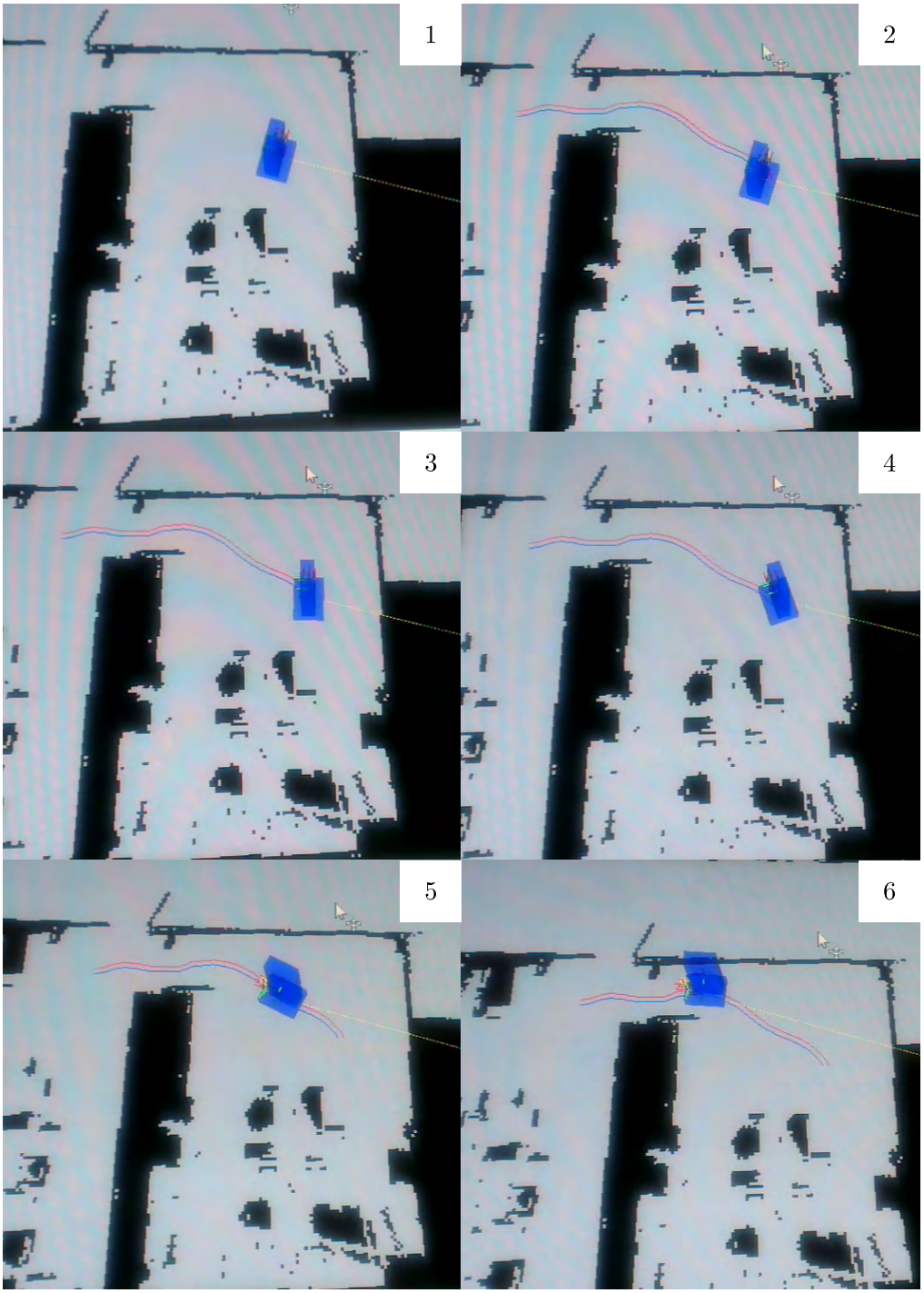

Figura 7.16: Diferentes ubicaciones del robot durante el recorrido de la trayectoria proporcionada por el planificador.

6. Calcular el término R_s.

7. Calcular el radio de giro efectivo, R_{sefec}, que permite al robot trazar un arco con el que llegar al punto destino seleccionado.

8. Calcular la velocidad final de desplazamiento lineal de la base, V_{bF}, a partir del radio de giro efectivo, R_{sefec}.

9. Calcular la velocidad final de desplazamiento lineal del motor de la rueda derecha e izquierda, V_{dF} y V_{iF}, a partir del término V_{bF}.

10. Calcular el tiempo que dura la mitad del perfil S de velocidad del motor de la rueda derecha e izquierda, T_d y T_i.

11. Comunicar los términos V_{dF}, V_{iF}, T_d y T_i a la PMAC a través de la DPRAM.

12. Esperar un periodo de tiempo de T_{MOV} ms para que el robot se mueva. Acabado este tiempo comenzar por el paso 1 de nuevo.

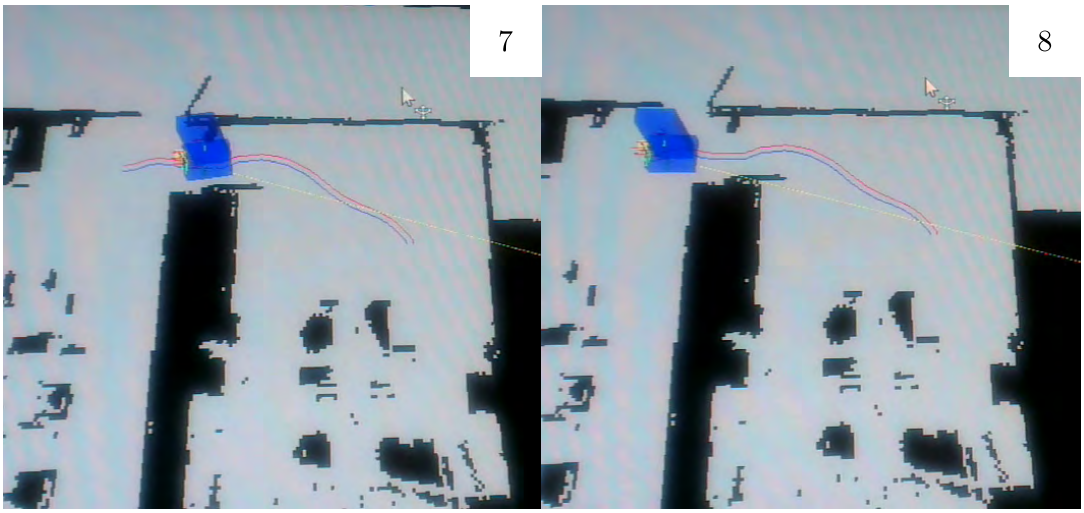

Figura 7.17: Diferentes ubicaciones del robot durante el recorrido de la trayectoria proporcionada por el planificador.

Si en el paso 4 el punto destino seleccionado es el último de la trayectoria de interés se calcula el último perfil S de velocidad de los motores. Una vez que estos los completen el robot se detendrá y el recorrido habrá finalizado. Tras el cálculo de θ_{err} y R_{sefec}, en los pasos 5 y 7 respectivamente,

se puede calcular el arco que debe recorrer cada rueda para detener el movimiento del robot en el último punto de la trayectoria de interés. El único elemento que resta por calcular es el tiempo que durará la mitad del perfil S de velocidad que se aplicará a cada rueda. A continuación se ejecuta el paso 11 de la lista anterior.

En la imagen 3, 4, 5 y 6 de la figura 7.16 se observar al robot en diferentes puntos de la trayectoria de interés. La imagen 6 de la figura 7.16 se corresponde con la fotografía 1 de la figura 7.18. En ambas imágenes se puede observar al robot acercándose a la puerta que une las dos salas del laboratorio 1.3.C13.

La imagen 7 de la figura 7.17 se corresponde con las fotografías 2 y 3 de la figura 7.18. En esas representaciones Manfred se haya cruzando el hueco de la puerta. Cabe destacar que el hueco de la puerta apenas es mayor que la anchura total del robot. Si el algoritmo pure-pursuit no consiguiese que el robot describiera la trayectoria de interés con mucha exactitud el sistema chocaría con el marco de la puerta. Por último la imagen 8 de la figura 7.17 se corresponde con la fotografía 4 de la figura 7.18. En ambas representación se observa que el robot Manfred ha cruzado con éxito el marco de la puerta y se dispone a encarar el tramo final de la ruta de interés.

Como se ha podido apreciar a lo largo de esta sección el algoritmo pure-pursuit implementado cumple con el objetivo de conseguir que el robot navegue de forma eficaz por trayectorias generadas por un planificador.

7.2. Conclusiones

Este trabajo nació con el objetivo de dotar al robot manipulador Manfred de una autonomía mayor de la que presentaba. Este proyecto final de carrera es la continuación natural del trabajo realizado en la tesis de máster [1].

En dicha tesis se elaboró un algoritmo de localización global basado en evolución diferencial. Este algoritmo de localización global es capaz de obtener la ubicación global del robot en el mapa del entorno usando tan sólo telemetría láser, es decir, la distancia del robot a los obstáculos situados a su alrededor, la odometría o medida del movimiento de las ruedas de la base y un mapa del entorno. Una vez que el robot es capaz de localizarse en un mapa el siguiente hito consiste en hacer que el robot pueda navegar de un punto a otro del entorno. Para conseguir este objetivo es necesario realizar multitud de tareas.

Figura 7.18: Robot Manfred cruzando el hueco de la puerta que comunica las dos salas que constituyen el laboratorio 1.3.C13.

En primer lugar se debe disponer de una arquitectura de control que permita el intercambio de información entre todos los módulos software presentes en el robot. De los múltiples frameworks disponibles en la actualidad que permiten construir una arquitectura de control para robots se eligió `ROS`, por ser un framework al alza, avalado por una gran comunidad de investigadores y desarrolladores, que dispone de muchas implementaciones de algoritmos, listas para ser usadas en tu robot.

A continuación se debe disponer de una algoritmo planificador de trayectorias que pueda construir la ruta óptima entre dos puntos cualesquiera del entorno. En este trabajo se ha usado el algoritmo de planificación de trayectorias denominado "fast marching", desarrollado en forma de librería C++ por Alberto Valero, profesor visitante del Departamento de Sistemas y Automática de la Universidad Carlos III .

Se disponía de un framework de control, una algoritmo de localización global y un algoritmo de planificación de trayectorias. Hacía falta elaborar un algoritmo de navegación que colaborase con los dos anteriores mediante el intercambio de información a través del framework `ROS`. De los numerosos tipos de algoritmos de navegación existentes quizás el más sencillo de implementar, el más popular y de los que mejores resultados proporciona sea el algoritmo pure-pursuit. Este algoritmo toma su nombre de una técnica de interceptación de aeronaves enemigas en combates aéreos.

El algoritmo pure-pursuit consiste en hacer que el robot trate de alcanzar, temporalmente, un punto destino situado en la trayectoria de interés a cierta distancia delante de él. Tras un breve lapso de tiempo en el que el robot se desplaza, tratando de alcanzar el punto objetivo indicado, el robot averigua su ubicación y a continuación elige un nuevo punto destino, de nuevo situado delante de él a cierta distancia. El algoritmo pure-pursuit sólo cuenta con un parámetro de control, la distancia a la que se encuentra el punto destino que persigue el robot. Cuando mayor sea esta distancia más suaviza el robot la trayectoria seguida, es decir, menos se ajusta el camino seguido a la ruta planificada. En cambio cuanto más pequeña sea esta distancia más se ciñe la trayectoria seguida a la trayectoria de interés, sin embargo aparece un efecto indeseable, que hace que la base esté continuamente corrigiendo su error de orientación para tratar de orientarse adecuadamente hacia la trayectoria. El efecto visual que produce en el observador es el de tener un robot que está dando bandazos a un lado y a otro de la ruta de interés.

Para encontrar un valor adecuado de la distancia se fijó un requisito más que razonable; conseguir que el robot se moviera siempre hacia delante, como los coches, de manera que la

velocidad de desplazamiento lineal de los motores de sus ruedas debería ser siempre positiva. Este requisito dio como resultado que la distancia a la que debe encontrarse el punto destino que persigue el robot tiene ser como mínimo igual al diámetro de la base de la plataforma robótica.

El algoritmo pure-pursuit implementado se ha dividido en tres fases principales. En la primera fase el robot se orienta hacia la trayectoria de interés, generada por el planificador, en caso de que el error de orientación sea superior a 5°. Una segunda fase, que se repite continuamente, en la que el robot se localiza en el mapa del entorno, selecciona el primer punto destino de la trayectoria de interés situado delante de él a una distancia mayor o igual al diámetro de la base y se mueve hacia el temporalmente, describiendo un arco que conecta ambos puntos. A continuación se inicia de nuevo la fase dos salvo que el punto destino seleccionado sea el último punto de la trayectoria de interés, momento en el que se inicia la tercera y última fase del algoritmo. En la tercera fase del algoritmo el robot va deteniéndose paulatinamente, finalizando su movimiento en el último punto de la trayectoria de interés. En la fase uno del algoritmo se controla a los motores de la base en posición, mientras que en las fases dos y tres se controlan en velocidad.

Para comprobar que el algoritmo desarrollado es eficaz se ha desarrollado un simulador en Matlab. De este modo se pueden hacer pruebas con el algoritmo antes de llevarlo a la plataforma real. En todas las pruebas realizadas y expuestas se ha podido ver el gran desempeño realizado por el algoritmo. Cabe destacar la última prueba, donde ha quedado patente que si el algoritmo no fuera capaz de seguir una trayectoria con suma exactitud el robot hubiera colisionado con el marco de la puerta.

Por último se resalta el gran esfuerzo que se ha realizado en este proyecto final de carrera por integrar todos los componentes software disponibles para el robot y hacerlos trabajar de manera coordinada.

7.3. Trabajo futuro

A continuación se expondrán algunas líneas de trabajo futuro relacionadas con proyecto final de carrera desarrollado en este documento. Como se ha podido comprobar durante la lectura del documento el algoritmo desarrollado es capaz de seguir una trayectoria de interés, generada por un planificador, entre la posición ocupada por el robot y un punto destino, elegido a priori y comunicado al algoritmo pure-pursuit al ejecutarlo.

Actualmente el robot no cuenta con la habilidad de modificar la trayectoria de interés ante la

aparición repentina de un obstáculo en el entorno por el que se desplaza. Por tanto la propuesta que se hace consiste dotar al robot de un mecanismo de actualización de ruta en caso de detectar un obstáculo que impida la realización de la ruta original.

El trabajo propuesto debería elaborar un módulo software de detección de obstáculos integrado en el framework `ROS`. Este módulo estaría suscrito a los mensajes publicados por el telémetro láser y a los mensajes publicados por el módulo de localización global por evolución diferencial. Además tendría acceso al mapa del entorno. De este modo dicho módulo podría estimar desde la ubicación del robot en el mapa del entorno las distancias a los obstáculos que tiene a su alrededor. Podría comparar estas distancias estimadas con las distancias reales tomadas por el telémetro láser del robot desde la ubicación ocupada en el entorno. Comparando estas distancias se puede averiguar si existen discrepancias entre el entorno que observa de verdad el robot y el entorno que espera encontrar en el mapa. Así se podrían identificar objetos presentes en el entorno no modelados en el mapa y objetos modelados en el mapa que han desaparecido del entorno. Estos obstáculos que aparecen y desaparecen del entorno pueden reflejarse dinámicamente sobre su mapa. Una vez que el mapa ha sido actualizado este módulo de software lo publica para que los nodos que lo necesiten puedan hacer uso de la actualización.

Por otro lado habría que modificar el módulo planificador de trayectorias existente. Actualmente este módulo es usado únicamente una vez al comienzo del algoritmo pure-pursuit para generar la trayectoria que conducirá al robot de su ubicación actual al punto destino de interés elegido. A día de hoy este módulo no tiene implementada la capacidad de modificar una trayectoria previa generada. Para que pueda modificarse la trayectoria de interés a seguir por el robot ante la aparición de un obstáculo se propone que este módulo se suscriba a los mensajes de actualización de mapa publicados por el módulo de detección de obstáculos. De este modo el planificador puede modificar, si fuera necesario, el fragmento de trayectoria de interés que se haya entre la ubicación actual ocupada por el robot y el punto destino de interés. La actualización de ruta es una tarea crítica y debe realizarse en algún punto de dicha trayectoria que aún no haya sido alcanzado por el robot pero que esté próximo el.

La segunda mejora que se le puede hacer al algoritmo pure-pursuit es ponderar el valor de la velocidad de desplazamiento lineal final de la base, V_{bF}, por un término que tenga en cuenta la proximidad de los obstáculos que tiene a su alrededor el robot. Actualmente el valor de dicha velocidad tiene en cuenta únicamente el radio de giro eficaz, R_{sefec}, que permite trazar un arco que une la posición del marco de referencia `link_base_desp` y el punto de control destino seleccionado

en la trayectoria de interés roja. Cuanto mayor es el valor del radio R_{sefec} mayor es el valor del término V_{bF}, siendo éste máximo cuando R_{sefec} es infinito, es decir, cuando el camino a seguir es una línea recta. Debido a la naturaleza del algoritmo de planificación usado, fast-marching, éste es capaz de proporcionar un coeficiente entre 0 y 1 por cada punto de la trayectoria de interés que indica, en tanto por ciento, cual debería ser la velocidad de desplazamiento lineal de la base, teniendo en cuenta la proximidad de los obstáculos que hay alrededor de la plataforma. Por tanto se tendría:

$$V_{bF_2} = \Gamma \cdot V_{bF} \tag{7.2}$$

siendo el término V_{bF} el que aparece en la expresión 6.14 del capítulo 6 y Γ el factor de corrección de la velocidad de desplazamiento lineal de la base que proporciona fast-marching. De esta manera la velocidad de desplazamiento lineal de la base tiene en cuenta tanto el radio de giro como la proximidad de los obstáculos situados alrededor del robot.

Con estas tareas se puede dotar a Manfred de un mayor grado de autonomía, permitiéndole reaccionar ante los obstáculos que aparezcan en el entorno por el que se mueve, ya sea mobiliario o personas.

Apéndice A

Perfiles de velocidad en la tarjeta PMAC

En cualquiera de los comandos que puede ejecutar la PMAC se puede especificar el perfil de velocidad que debe seguir el motor durante su movimiento. La PMAC ofrece mucha flexibilidad a la hora de indicar cómo deber ser este perfil de velocidad. Es necesario entender adecuadamente estos perfiles para poder desarrollar estrategias eficaces de movimientos para el robot en aras de conseguir un desplazamiento completamente autónomo. Así, cuando el robot deba recorrer una distancia específica será adecuado comandar al robot en posición, mientras que si la distancia a recorrer no es el factor crítico, y sí lo es la trayectoria o distancia de seguridad a los obstáculos cercanos será interesante comandar al robot en velocidad. A continuación se realiza un exhaustivo desarrollo matemático de los diferentes perfiles de velocidad que puede ejecutar la PMAC.

La función *velocidad de desplazamiento lineal* en una rueda motriz cuando se usa un comando `pvt` o un comando `jog` en la PMAC es una función cuadrática que se desarrolla durante un intervalo de tiempo T.

$$t_0 \leq t \leq t_0 + T$$

$$v(t) = A \cdot (t - t_0)^2 + B \cdot (t - t_0) + C \tag{A.1}$$

Para definir esta expresión es necesario conocer el valor de los coeficientes A, B y C. Para hallar estos coeficientes es necesario conocer el valor de la función en tres instantes distintos.

Estas tres condiciones iniciales son:

$$V_I = v(t_0) \tag{A.2}$$

$$V_M = v\left(t_0 + \frac{T}{2}\right) \tag{A.3}$$

$$V_F = v(t_0 + T) \tag{A.4}$$

Conocidas las condiciones iniciales se pueden despejar en la expresión A.1 el valor de los coeficientes A, B y C.

$$C = V_I \tag{A.5}$$

$$B = \frac{-V_F + 4 \cdot V_M - 3 \cdot V_I}{T} \tag{A.6}$$

$$A = 2 \cdot \frac{V_F - 2 \cdot V_M + V_I}{T^2} \tag{A.7}$$

Dado que la función aceleración es la derivada de la función velocidad, ésta primera será una expresión lineal:

$$a(t) = \frac{\partial v}{\partial t} = 2 \cdot A \cdot (t - t_0) + B \tag{A.8}$$

$$A_I = a(t_0) = B \tag{A.9}$$

$$A_M = a\left(t_0 + \frac{T}{2}\right) = A \cdot T + B = \frac{V_F - V_I}{T} \tag{A.10}$$

$$A_F = a(t_0 + T) = 2 \cdot A \cdot T + B = \frac{3 \cdot V_F - 4 \cdot V_M + V_I}{T} \tag{A.11}$$

La función desplazamiento es la integral de la función velocidad, por tanto será una expresión cúbica:

$$l(t) = \int_{t_0}^{t} v(\tau) \cdot d\tau + L_I \tag{A.12}$$

$$= \frac{A}{3} \cdot (t - t_0)^3 + \frac{B}{2}(t - t_0)^2 + C \cdot (t - t_0) + L_I \tag{A.13}$$

donde el término L_I es el desplazamiento inicial de la rueda con respecto a un lugar de referencia conocido.

$$L_I = l(t_0) \tag{A.14}$$

$$L_M = l\left(t_0 + \frac{T}{2}\right) = \frac{T}{24} \cdot (-V_F + 8 \cdot V_M + 5 \cdot V_I) + L_I \tag{A.15}$$

$$L_F = l(t_0 + T) = \frac{T}{6} \cdot (V_F + 4 \cdot V_M + V_I) + L_I \tag{A.16}$$

Finalmente, el recorrido realizado por la rueda entre el instante t_0 y $t_0 + \frac{T}{2}$, entre el instante $t_0 + \frac{T}{2}$ y $t_0 + T$ y entre el instante t_0 y $t_0 + T$ es respectivamente:

$$\Delta L_1 = L_M - L_I = \frac{T}{24} \cdot (-V_F + 8 \cdot V_M + 5 \cdot V_I) \tag{A.17}$$

$$\Delta L_2 = L_F - L_M = \frac{T}{24} \cdot (5 \cdot V_F + 8 \cdot V_M - V_I) \tag{A.18}$$

$$\Delta L_3 = L_F - L_I = \frac{T}{6} \cdot (V_F + 4 \cdot V_M + V_I) \tag{A.19}$$

A.1. Perfil de velocidad tipo 0

Se denomina perfil de velocidad de tipo 0 a la función de velocidad de desplazamiento que se obtiene cuando las condiciones iniciales son:

$$V_I = V_M = V_F \tag{A.20}$$

$$A = B = 0 \tag{A.21}$$

$$C = V_I \tag{A.22}$$

$$v(t) = C \tag{A.23}$$

$$A_I = A_M = A_F = a(t) = 0 \tag{A.24}$$

$$l(t) = C \cdot (t - t_0) + L_I \tag{A.25}$$

$$L_M = \frac{T}{2} \cdot V_I + L_I \tag{A.26}$$

$$L_F = T \cdot V_I + L_I \tag{A.27}$$

$$\Delta L_1 = \frac{T}{2} \cdot V_I \tag{A.28}$$

$$\Delta L_2 = \frac{T}{2} \cdot V_I \tag{A.29}$$

$$\Delta L_3 = T \cdot V_I \tag{A.30}$$

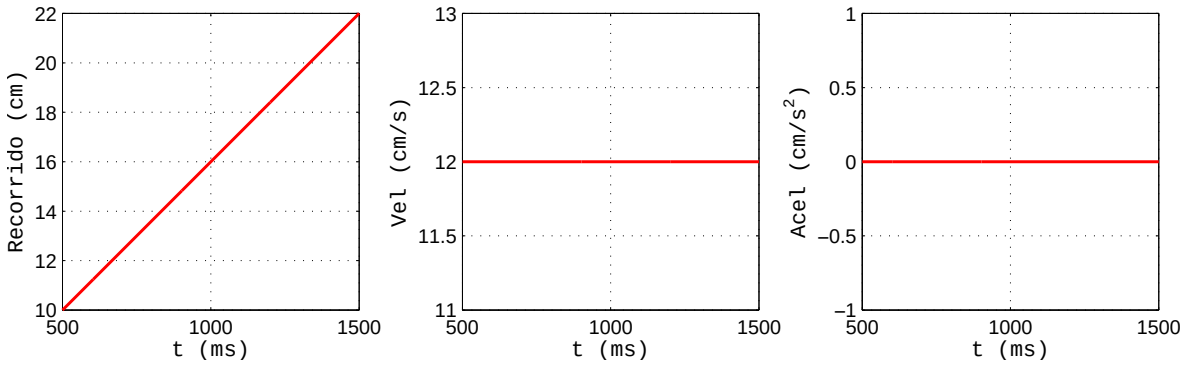

Figura A.1: Perfil de velocidad de tipo 0. $V_I = 12\,\text{cm/s}$. $L_I = 10\,\text{cm}$. $T = 1000\,\text{ms}$.

A.2. Perfil de velocidad tipo 1

Se denomina perfil de velocidad de tipo 1 a la función de velocidad de desplazamiento que se obtiene cuando las condiciones iniciales son:

$$V_I \neq V_F \tag{A.31}$$

$$V_M = \frac{V_F + V_I}{2} \tag{A.32}$$

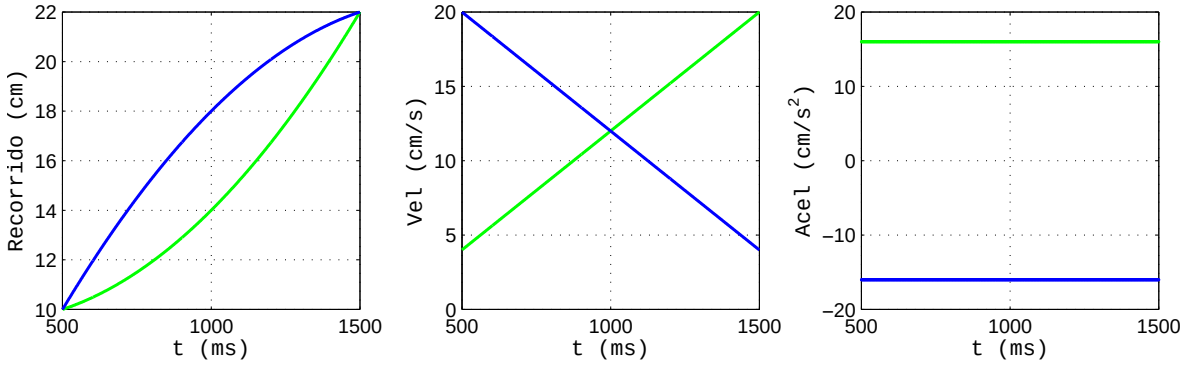

Figura A.2: Perfil de velocidad de tipo 1. En verde perfil 1 ascendente. En azul perfil 1 descendente $V_I = 4\,\text{cm/s}$. $V_F = 20\,\text{cm/s}$. $L_I = 10\,\text{cm}$. $T = 1000\,\text{ms}$.

Con los valores anteriores de las condiciones iniciales el perfil de velocidad de tipo 1 queda

de la forma:

$$A = 0 \tag{A.33}$$

$$B = \frac{V_F - V_I}{T} \tag{A.34}$$

$$C = V_I \tag{A.35}$$

$$v(t) = B \cdot (t - t_0) + C \tag{A.36}$$

En estas circunstancias la aceleración es una función constante y el desplazamiento es una función cuadrática:

$$A_I = A_M = A_F = a(t) = B \tag{A.37}$$

$$l(t) = \frac{B}{2} \cdot (t - t_0)^2 + C \cdot (t - t_0) + L_I \tag{A.38}$$

$$L_M = \frac{T}{8} \cdot (V_F + 3 \cdot V_I) + L_I \tag{A.39}$$

$$L_F = \frac{T}{2} \cdot (V_F + V_I) + L_I \tag{A.40}$$

$$\Delta L_1 = \frac{T}{8} \cdot (V_F + 3 \cdot V_I) \tag{A.41}$$

$$\Delta L_2 = \frac{T}{8} \cdot (3 \cdot V_F + V_I) \tag{A.42}$$

$$\Delta L_3 = \frac{T}{2} \cdot (V_F + V_I) \tag{A.43}$$

A.3. Perfil de velocidad tipo 2

Se denomina perfil de velocidad de tipo 2 a la función de velocidad de desplazamiento cuadrática que tiene su vértice en las coordenadas:

$$\left(x_{\text{vértice}}, y_{\text{vértice}}\right) = (t_0, V_I)$$

y cuyas condiciones iniciales son:

$$V_I \neq V_F$$

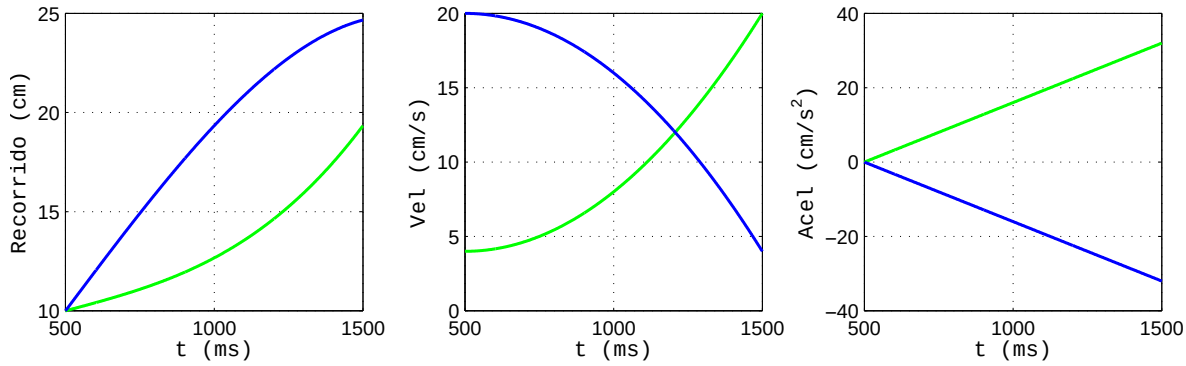

Figura A.3: Perfil de velocidad de tipo 2. En verde perfil 2 ascendente. En azul perfil 2 descendente $V_I = 4\,\text{cm/s}$. $V_F = 20\,\text{cm/s}$. $L_I = 10\,\text{cm}$. $T = 1000\,\text{ms}$.

Para una función cuadrática que tiene su vértice en el punto anterior, el valor del coeficiente A puede hallarse a partir de las expresiones:

$$4 \cdot a \cdot \left(v - y_{\text{vértice}}\right) = \left(t - x_{\text{vértice}}\right)^2 \tag{A.44}$$

$$A = \frac{1}{4 \cdot a} \tag{A.45}$$

Por tanto:

$$4 \cdot a \cdot \left(v - V_I\right) = \left(t - t_0\right)^2 \tag{A.46}$$

$$A = \frac{V_F - V_I}{T^2} \tag{A.47}$$

Conociendo el valor del coeficiente A y del coeficiente C

$$C = V_I$$

se puede obtener el valor de la velocidad V_M a partir de la expresión A.7.

$$V_M = \frac{1}{4} \cdot \left(V_F + 3 \cdot V_I\right) \tag{A.48}$$

Conociendo el valor del término V_M en función de los términos V_I y V_F se obtiene el valor del coeficiente B a partir de la expresión A.6.

$$B = 0 \tag{A.49}$$

Conocidos estos términos se puede hallar la expresión de la función velocidad de desplazamiento, de la función aceleración y de la función desplazamiento:

$$v\,(t) \;=\; A \cdot \,(t \,-\, t_0)^2 \,+\, C \tag{A.50}$$

$$a\,(t) \;=\; 2 \cdot A\,(t \,-\, t_0) \tag{A.51}$$

$$A_I \;=\; 0 \tag{A.52}$$

$$A_M \;=\; \frac{V_F \,-\, V_I}{T} \tag{A.53}$$

$$A_F \;=\; 2 \cdot A_M \tag{A.54}$$

$$l\,(t) \;=\; \frac{A}{3} \cdot \,(t \,-\, t_0)^3 \,+\, C \cdot \,(t \,-\, t_0) \,+\, L_I \tag{A.55}$$

$$L_M \;=\; \frac{T}{24} \cdot \,(V_F \,+\, 11 \cdot V_I) \,+\, L_I \tag{A.56}$$

$$L_F \;=\; \frac{T}{3} \cdot \,(V_F \,+\, 2 \cdot V_I) \,+\, L_I \tag{A.57}$$

$$\Delta L_1 \;=\; \frac{T}{24} \cdot \,(V_F \,+\, 11 \cdot V_I) \tag{A.58}$$

$$\Delta L_2 \;=\; \frac{T}{24} \cdot \,(7 \cdot V_F \,+\, 5 \cdot V_I) \tag{A.59}$$

$$\Delta L_3 \;=\; \frac{T}{3} \cdot \,(V_F \,+\, 2 \cdot V_I) \tag{A.60}$$

A.4. Perfil de velocidad tipo 3

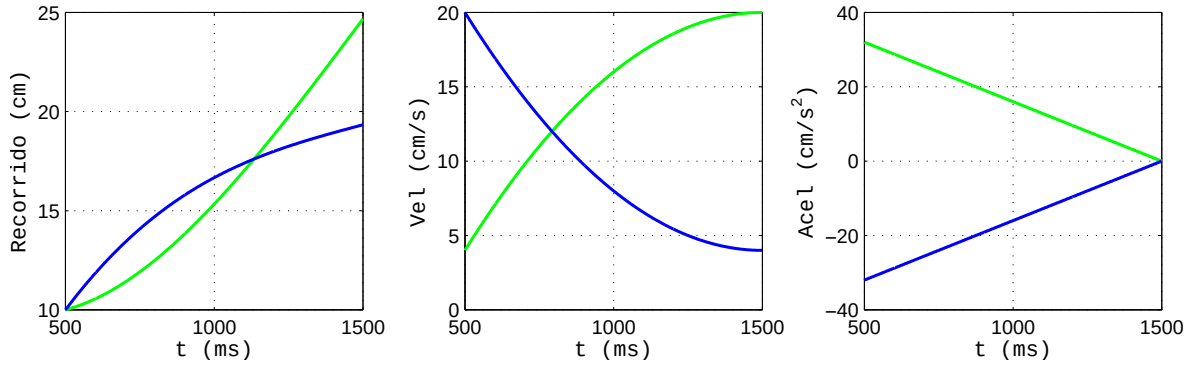

Figura A.4: Perfil de velocidad de tipo 3. En verde perfil 3 ascendente. En azul perfil 3 descendente $V_I \;=\; 4\,\text{cm/s}$. $V_F \;=\; 20\,\text{cm/s}$. $L_I \;=\; 10\,\text{cm}$. $T \;=\; 1000\,\text{ms}$.

Se denomina perfil de velocidad de tipo 3 a la función de velocidad de desplazamiento cuadrática que tiene su vértice en las coordenadas:

$$\left(x_{\text{vértice}}, y_{\text{vértice}}\right) = (t_0 + T, V_F)$$

y cuyas condiciones iniciales son:

$$V_I \neq V_F$$

Con estos datos se puede hallar el valor del coeficiente A:

$$4 \cdot a \cdot (v - V_F) = (t - (t_0 + T))^2 \tag{A.61}$$

$$A = \frac{1}{4 \cdot a} = -\left(\frac{V_F - V_I}{T^2}\right) \tag{A.62}$$

Del mismo modo que se hizo en el perfil de velocidad de tipo 2, conociendo el valor del coeficiente A y del coeficiente C

$$C = V_I$$

se puede obtener el valor de la velocidad V_M a partir de la expresión A.7.

$$V_M = \frac{1}{4} \cdot (3 \cdot V_F + V_I) \tag{A.63}$$

Conociendo el valor del término V_M en función de los términos V_I y V_F se obtiene el valor del coeficiente B a partir de la expresión A.6.

$$B = 2 \cdot \left(\frac{V_F - V_I}{T}\right) \tag{A.64}$$

Conociendo el valor de todos estos términos se pueden obtener las expresiones de las funciones velocidad, aceleración y desplazamiento:

$$v(t) = A \cdot (t - (t_0 + T))^2 + V_F \tag{A.65}$$

$$a(t) = 2 \cdot A\,(t - (t_0 + T)) \tag{A.66}$$

$$A_I = B \tag{A.67}$$

$$A_M = \frac{B}{2} \tag{A.68}$$

$$A_F = 0 \tag{A.69}$$

$$l\,(t) \;=\; \frac{A}{3}\cdot\,(t\,-\,(t_0\,+\,T))^3\,+\,V_F\cdot\,(t\,-\,t_0)\,-\,\frac{T}{3}\cdot\,(V_F\,-\,V_I)\,+\,L_I \tag{A.70}$$

$$L_M \;=\; \frac{T}{24}\cdot\,(5\cdot V_F\,+\,7\cdot V_I)\,+\,L_I \tag{A.71}$$

$$L_F \;=\; \frac{T}{3}\cdot\,(2\cdot V_F\,+\,V_I)\,+\,L_I \tag{A.72}$$

$$\Delta L_1 \;=\; \frac{T}{24}\cdot\,(5\cdot V_F\,+\,7\cdot V_I) \tag{A.73}$$

$$\Delta L_2 \;=\; \frac{T}{24}\cdot\,(11\cdot V_F\,+\,V_I) \tag{A.74}$$

$$\Delta L_3 \;=\; \frac{T}{3}\cdot\,(2\cdot V_F\,+\,V_I) \tag{A.75}$$

A.5. Perfil de velocidad tipo 4

Se denomina perfil de velocidad de tipo 4 a la función de velocidad de desplazamiento que se obtiene concatenando un perfil de velocidad de tipo 2 seguido de un perfil de velocidad de tipo 3. La duración de ambos perfiles de velocidad es T ms, por tanto el movimiento completo dura $2 \cdot$ T ms.

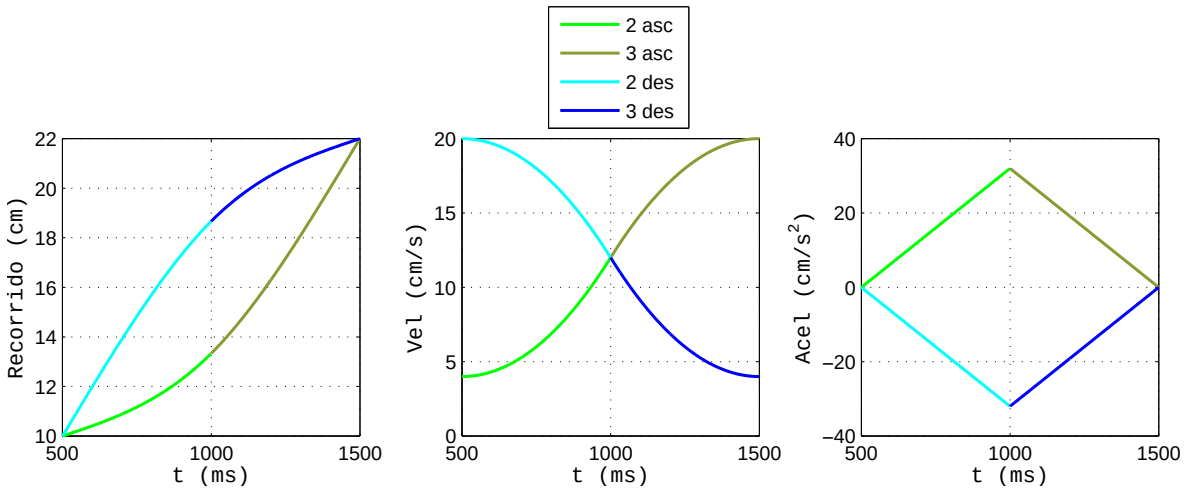

Figura A.5: Perfil de velocidad de tipo 4. En verde, perfil 2 ascendente seguido de perfil 3 ascendente. En azul, perfil 2 descendente seguido de perfil 3 descendente. $V_I = 4$ cm/s. $V_F = 20$ cm/s. $L_I = 10$ cm. $T = 500$ ms.

Conviene definir la nomenclatura usada en este último caso. Todos los términos que incluyen el subíndice 1 están vinculados al primero de los dos perfiles de velocidad que constituyen el perfil

de velocidad de tipo 4; siendo el primero de los perfiles de tipo 2. Así V_{I_1} es el valor inicial del perfil de velocidad de tipo 2; es decir, en el instante t_0. Por otro lado los términos que incluyen el subíndice 2 están vinculados al segundo de los dos perfiles de velocidad que constituyen el perfil de velocidad de tipo 4; siendo este segundo perfil de tipo 3. Así V_{I_2} es el valor inicial del perfil de velocidad de tipo 3; es decir, en el instante $t_0 + T$.

El término V_I es el valor inicial del perfil de velocidad de tipo 4; es decir, en el instante t_0, por lo cual coincide con el valor del término V_{I_1}. El término V_F es el valor final del perfil de velocidad de tipo 4, por lo cual coincide con el valor final del perfil de velocidad de tipo 3, V_{F_2}. El valor a la mitad del perfil de velocidad de tipo 4, V_M, coincide con el valor final del perfil de velocidad de tipo 2, V_{F_1} y con el valor inicial del perfil de velocidad de tipo 3, V_{I_2}.

$$V_I = V_{I_1} \tag{A.76}$$

$$V_M = V_{F_1} = V_{I_2} \tag{A.77}$$

$$V_F = V_{F_2} \tag{A.78}$$

El perfil de velocidad de tipo 4 cumple con la condición inicial:

$$V_M = \frac{V_F + V_I}{2} \tag{A.79}$$

A.5.1. Primera mitad: perfil de velocidad de tipo 2

La primera mitad del perfil de velocidad de tipo 4 está constituido por un perfil de velocidad de tipo 2. Para obtener los términos asociados al perfil de velocidad de tipo 2 se usan las expresiones A.44 a A.60, A.76, A.78 y teniendo en cuenta que:

$$V_{F_1} = V_{I_2} = V_M = \frac{V_F + V_I}{2} \tag{A.80}$$

$$t_0 \leq t \leq t_0 + T$$

$$A_1 = \frac{V_{F_1} - V_{I_1}}{T^2} = \frac{V_F - V_I}{2 \cdot T^2} \tag{A.81}$$

$$B_1 = 0 \tag{A.82}$$

$$C_1 = V_{I_1} = V_I \tag{A.83}$$

$$v_1(t) = A_1 \cdot (t - t_0)^2 + C_1 \tag{A.84}$$

$$V_{M1} = \frac{1}{4} \cdot (V_{F_1} + 3 \cdot V_{I_1}) = \frac{1}{8} \cdot (V_F + 7 \cdot V_I) \tag{A.85}$$

$$a_1(t) = 2 \cdot A_1 \cdot (t - t_0) \tag{A.86}$$

$$A_I = A_{I_1} = 0 \tag{A.87}$$

$$A_{M1} = \frac{V_F - V_I}{2 \cdot T} \tag{A.88}$$

$$A_M = A_{F_1} = A_{I_2} = \frac{V_F - V_I}{T} \tag{A.89}$$

$$l_1(t) = \frac{A_1}{3} \cdot (t - t_0)^3 + C_1 \cdot (t - t_0) + L_I \tag{A.90}$$

$$L_{M1} = \frac{T}{48} \cdot (V_F + 23 \cdot V_I) + L_I \tag{A.91}$$

$$L_M = L_{F_1} = L_{I_2} = \frac{T}{6} \cdot (V_F + 5 \cdot V_I) + L_I \tag{A.92}$$

$$\Delta L_{11} = L_{M1} - L_I = \frac{T}{48} \cdot (V_F + 23 \cdot V_I) \tag{A.93}$$

$$\Delta L_{21} = L_{F_1} - L_{M1} = L_M - L_{M1} = \frac{T}{48} \cdot (7 \cdot V_F + 17 \cdot V_I) \tag{A.94}$$

$$\Delta L_1 = \Delta L_{31} = L_{F_1} - L_{I_1} = L_M - L_I = \frac{T}{6} \cdot (V_F + 5 \cdot V_I) \tag{A.95}$$

A.5.2. Segunda mitad: perfil de velocidad de tipo 3

La segunda mitad del perfil de velocidad de tipo 4 está constituido por un perfil de velocidad de tipo 3. Para obtener los términos asociados al perfil de velocidad de tipo 3 se usan las expresiones A.61 a A.75, A.76, A.78 y A.80. En la expresión A.61 se debe tener en cuenta que el punto donde se localiza el vértice del perfil de velocidad de tipo 3 es:

$$(x_{\text{vértice}}, y_{\text{vértice}}) = (V_F, t_0 + 2 \cdot T)$$

$$A_2 = -\left(\frac{V_F - V_{I_2}}{T^2}\right) = -\left(\frac{V_F - V_I}{2 \cdot T^2}\right) = -A_1 \tag{A.96}$$

$$B_2 = 2 \cdot \frac{V_F - V_{I_2}}{T} = \frac{V_F - V_I}{T} \tag{A.97}$$

$$C_2 = V_{I_2} = V_M = \frac{V_F + V_I}{2} \tag{A.98}$$

$$v_2(t) = A_2 \cdot (t - (t_0 + 2 \cdot T))^2 + V_F \tag{A.99}$$

$$V_{M2} = \frac{1}{4} \cdot (3 \cdot V_{F_2} + V_{I_2}) = \frac{1}{8} \cdot (7 \cdot V_F + V_I) \tag{A.100}$$

$$a_2(t) = 2 \cdot A_2 \cdot (t - (t_0 + 2 \cdot T)) \tag{A.101}$$

$$A_M = A_{I_2} = A_{F_1} = B_2 \tag{A.102}$$

$$A_{M2} = \frac{B_2}{2} \tag{A.103}$$

$$A_F = A_{F_2} = 0 \tag{A.104}$$

$$l_2(t) = \frac{A_2}{3} \cdot (t - (t_0 + 2 \cdot T))^3 + V_F \cdot (t - (t_0 + T)) + V_I \cdot T + L_I \tag{A.105}$$

$$L_M = L_{I_2} = L_{F_1} = \frac{T}{6} \cdot (V_F + 5 \cdot V_I) + L_I \tag{A.106}$$

$$L_{M2} = \frac{T}{48} \cdot (25 \cdot V_F + 47 \cdot V_I) + L_I \tag{A.107}$$

$$L_F = L_{F_2} = T \cdot (V_F + V_I) + L_I \tag{A.108}$$

$$\Delta L_{12} = L_{M2} - L_{I_2} = L_{M2} - L_M = \frac{T}{48} \cdot (17 \cdot V_F + 7 \cdot V_I) \tag{A.109}$$

$$\Delta L_{22} = L_{F_2} - L_{M2} = L_F - L_{M2} = \frac{T}{48} \cdot (23 \cdot V_F + V_I) \tag{A.110}$$

$$\Delta L_2 = \Delta L_{32} = L_{F_2} - L_{I_2} = L_F - L_M = \frac{T}{6} \cdot (5 \cdot V_F + V_I) \tag{A.111}$$

$$\Delta L_3 = L_{F_2} - L_{I_1} = L_F - L_I = T \cdot (V_F + V_I) \tag{A.112}$$

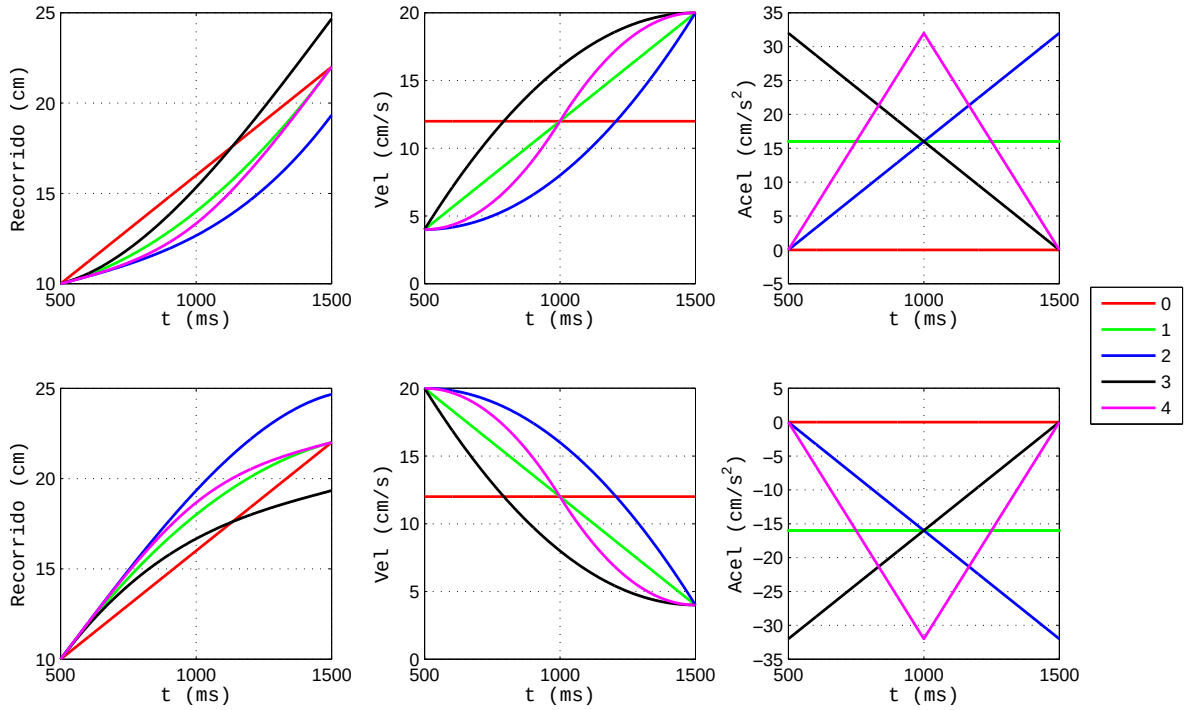

Figura A.6: Comparación de perfiles de velocidad. Para perfil 0, $V_I = V_M = V_F = 12\,\text{cm/s}$. Para perfiles 1, 2, 3 y 4, $V_I = 4\,\text{cm/s}$, $V_F = 20\,\text{cm/s}$, $L_I = 10\,\text{cm}$. Para perfiles 0, 1, 2 y 3, $T = 1000\,\text{ms}$. Para perfil 4, $T = 500\,\text{ms}$.

El perfil de velocidad usado siempre en los motores de las ruedas motrices es el perfil número 4, ya que asegura una aceleración continua durante todo el intervalo de tiempo que dura el movimiento.

Se ha analizado la velocidad de desplazamiento lineal de las ruedas motrices; sin embargo, se puede hacer el mismo análisis desde la perspectiva de la velocidad angular de las mismas sin más que tener en cuenta la siguiente relación:

$$\omega\left(t\right) = \frac{v\left(t\right)}{r_r} \tag{A.113}$$

donde el término r_r representa el radio de las ruedas motrices.

El análisis matemático desarrollado en este capítulo es válido para el resto de motores del robot (motores del brazo). En ese caso, es más adecuado realizar el análisis desde la perspectiva de su velocidad angular.

Bibliografía

[1] J. F. R. Crespo, "Localización global 2D de robots móviles usando evolución diferencial dentro del framework ROS," Master's thesis, Universidad Carlos III de Madrid, 2012.

[2] J. V. Girbés, "Generación de trayectorias de curvatura continua para el seguimiento de líneas basado en visión artificial," Master's thesis, Universidad de Valencia, 2010.

[3] A. B. Azcón, "Análisis y diseño del control de posición de un robot móvil con tracción diferencial," Master's thesis, Universitat Rovira I Virgili, 2003.

[4] R. Goebel, *ROS by example.* Independently published, 2013.

[5] A. Martínez and E. Fernández, *Learning ROS for robotics programming.* Packt Publishing, 2013.

[6] A. O. Baturone, *Robótica: Manipuladores y robots móviles.* Marcombo, S.A, 2001.

[7] J. M. O'Kane, *A gentle introduction to ROS.* Independently published, 2013.

[8] C. Lewin, *Mathematics of motion control profiles.* Performance Motion Devices, 2009.

[9] R. Coulter, "Implementation of the pure pursuit path tracking algorithm," Master's thesis, Carnegie Mellon University, 1992.

[10] O. Amidi, "Integrated mobile robot control," Master's thesis, Carnegie Mellon University, 1990.

[11] J. Giesbrecht, D. Mackay, J. Collier, and S. Verret, "Path tracking for unmanned ground vehicle navigation," tech. rep., Defence Research and Development Canada, 2005.

[12] J. M.Snider, "Automatic steering methods for autonomous automobile path tracking," Master's thesis, Carnegie Mellon University, 2009.

[13] G. Lucas, "A path following a circular arc to a point at a specified range and bearing." http://rossum.sourceforge.net/papers/CalculationsForRobotics/CirclePath.htm, 2006. Accessed September 20, 2013.

[14] J. A. Valencia, A. M. Oregón, and L. H. Ríos, "Modelo cinemático de un robot móvil tipo diferencial y navegación a partir de la estimación odométrica," *Scientia et Technica*, no. 41, 2009.

[15] M. Lundgren, "Path tracking and obstacle avoidance for a miniature robot," Master's thesis, Umea University, 2003.

[16] S. F.Campbell, "Steering control of an autonomous ground vehicle with application to the DARPA urban challenge," Master's thesis, Massachusetts Institute of Technology, 2007.

[17] M. Gauland and G. Lucas, "A tutorial and elementary trajectory model for the differential steering system of robot wheel actuators." http://rossum.sourceforge.net/papers/DiffSteer, 2006. Accessed September 18, 2013.

[18] M. Berkson, "Finding differential wheel velocity as a function of turn angle and radius and constant acceleration." http://rossum.sourceforge.net/papers/CalculationsForRobotics/DifferentialWheelVelocity/index.htm, 2006. Accessed August 15, 2013.

[19] G. Lucas, "A path based on third-degree polynomials, constrained at its endpoints by position, course, and speed." http://rossum.sourceforge.net/papers/CalculationsForRobotics/CubicPath.htm, 2006. Accessed August 17, 2013.

[20] G. Lucas, "A compendium to calculations useful for robotics." http://rossum.sourceforge.net/papers/CalculationsForRobotics, 2006. Accessed August 25, 2013.

[21] D. Goma, *Manual de aplicación de encoders*. West Instruments de Mexico, S.A, 2006.

[22] C. Eltra, "Encoder incremental: Descripción general." http://facultad.bayamon.inter.edu/arincon/encoderincrementales.pdf, 2000. Accedido Diciembre 30, 2013.

[23] M. Carmona, "El encoder." http://www.mcbtec.com/Funcionamiento_Encoder.pdf, 2008. Accedido Diciembre 25, 2013.

[24] J. A. Tello-Cristiany, R. Silva-Ortigoza, and M. Marciano-Melchor, "Desarrollo del modelo cinemático de un RMR a partir de las ecuaciones odométricas," *Latin-American Journal of Physics Education*, 2011.

[25] C. F. Caramés, "Lasap: Desarrollo e implementación de utilidades de SLAM para un robot móvil utilizando un dispositivo LMS," Master's thesis, Universidad de Salamanca, 2005.

[26] A. Blomdell, *JR3: Installation manual for force-torque sensors with internal electronics*. JR3, Inc, 2013.

[27] S. A. Ansari, *Desarrollo de un Manipulador Móvil Autónomo con Características Antropométricas*. PhD thesis, Universidad Carlos III de Madrid, 2010.

[28] B. Pedersen, *PMAC - Dual Ported Ram*. Delta Tau Data Systems, Inc., 2004.

[29] B. Pedersen, *PMAC2: Software manual*. Delta Tau Data Systems, Inc., 2008.

[30] C. G. Mori, *PMAC2: User manual*. Delta Tau Data Systems, Inc., 2007.

[31] F. J. Á. Fernández, "Desarrollo de los drivers para la tarjeta controladora PMAC2-PCI para el manipulador móvil Manfred," Master's thesis, Universidad Carlos III de Madrid, 2007.

[32] D. Blanco, S. A. Ansari, C. Castejón, B. L. Boada, and L. E. Moreno, "Manfred: Robot antropomórfico de servicios fiable y seguro para operar en entornos humanos," *Revista Iberoamericana de Ingeniería Mecánica*, vol. 9, no. 3, pp. 33 – 48, 2005.